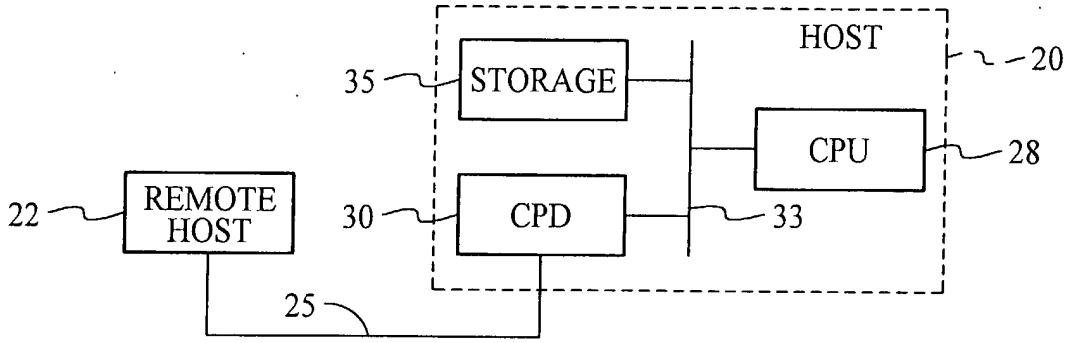

HOST
20
35
STORAGE
CPU
28
22
REMOTE HOST
30
CPD
33
25

FIG. 1

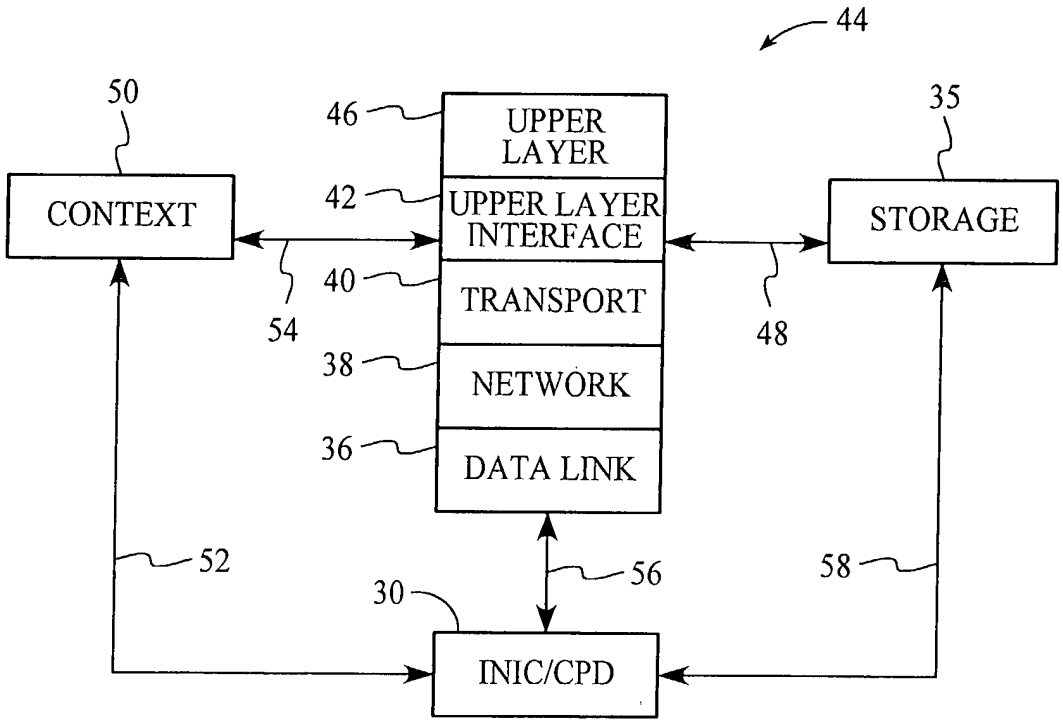

44
50
46
UPPER LAYER
35
CONTEXT
42
UPPER LAYER INTERFACE
STORAGE
54
40
TRANSPORT
48
38
NETWORK
36
DATA LINK
52
56
58
30
INIC/CPD

FIG. 2

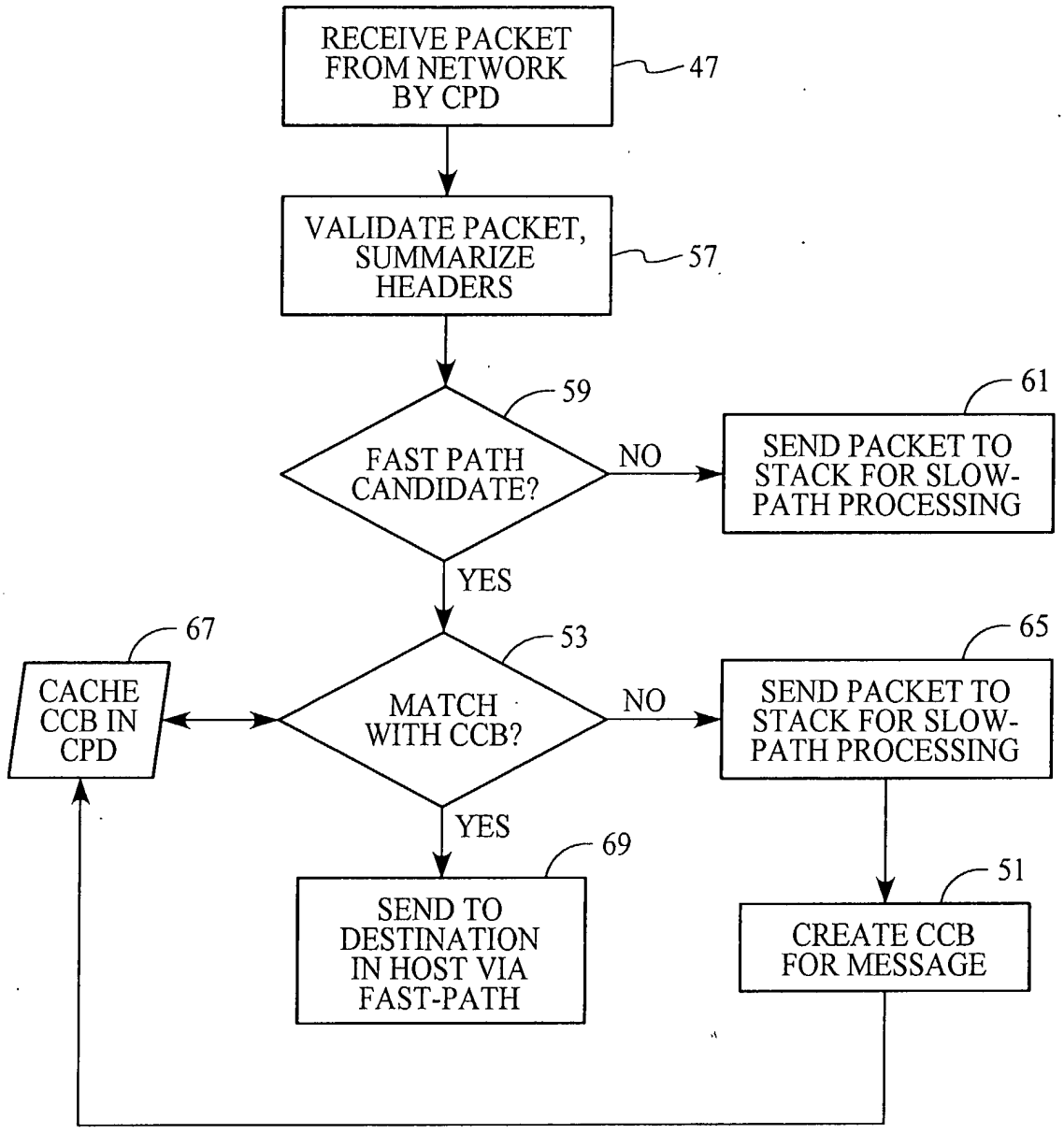

RECEIVE PACKET FROM NETWORK BY CPD
47
VALIDATE PACKET, SUMMARIZE HEADERS
57
59
FAST PATH CANDIDATE?
NO
61
SEND PACKET TO STACK FOR SLOW-PATH PROCESSING
YES
67
CACHE CCB IN CPD
53
MATCH WITH CCB?
NO
65
SEND PACKET TO STACK FOR SLOW-PATH PROCESSING
YES
69
SEND TO DESTINATION IN HOST VIA FAST-PATH
51
CREATE CCB FOR MESSAGE

FIG. 3

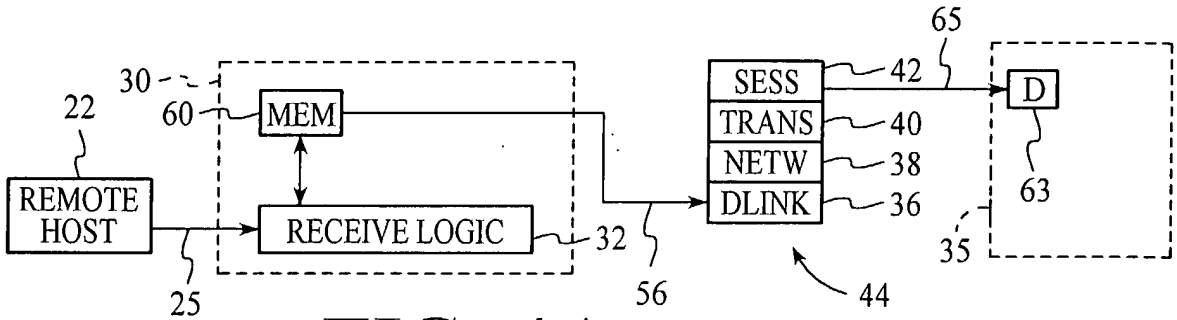


FIG. 4A

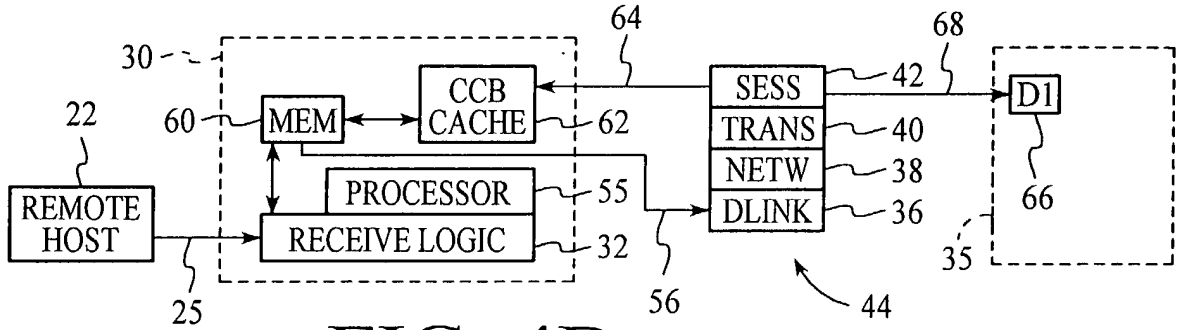


FIG. 4B

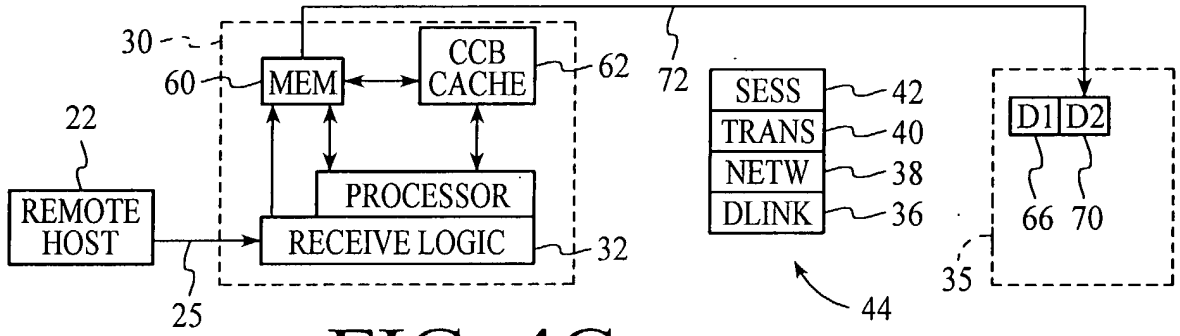


FIG. 4C

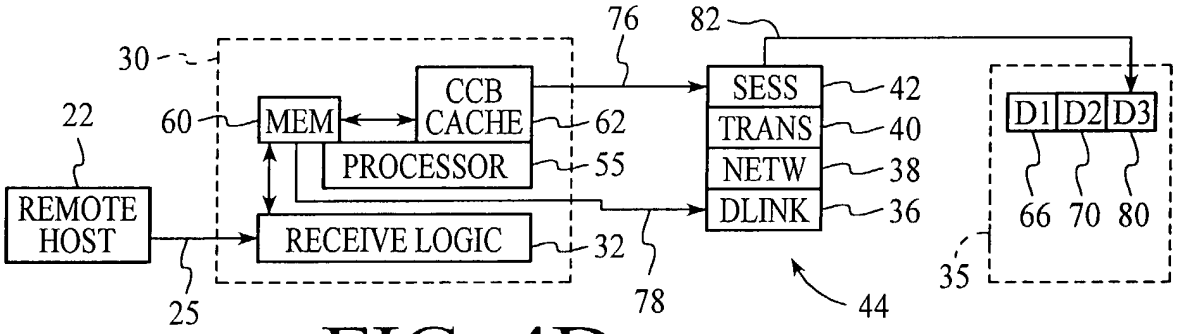


FIG. 4D

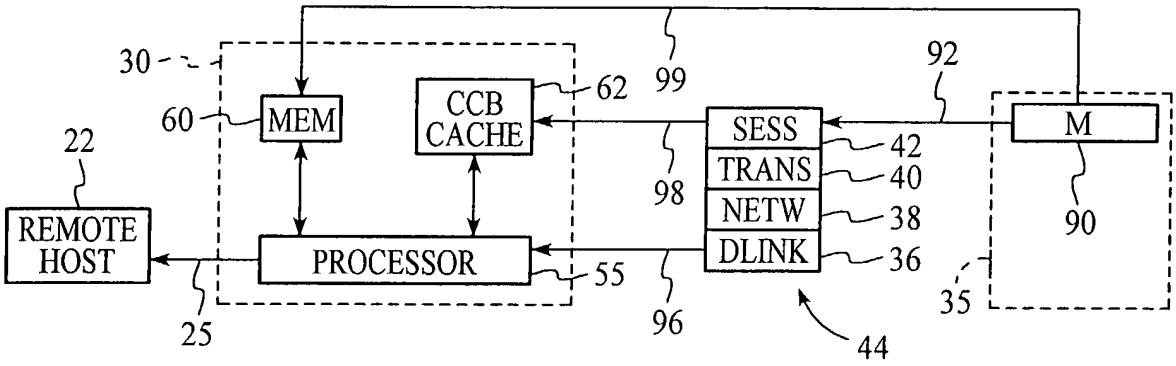

30
60
MEM
CCB CACHE
62
99
92
22
REMOTE HOST
PROCESSOR
55
98
SESS
TRANS
NETW
DLINK
42
40
38
36
M
90
35
25
96
44

FIG. 5

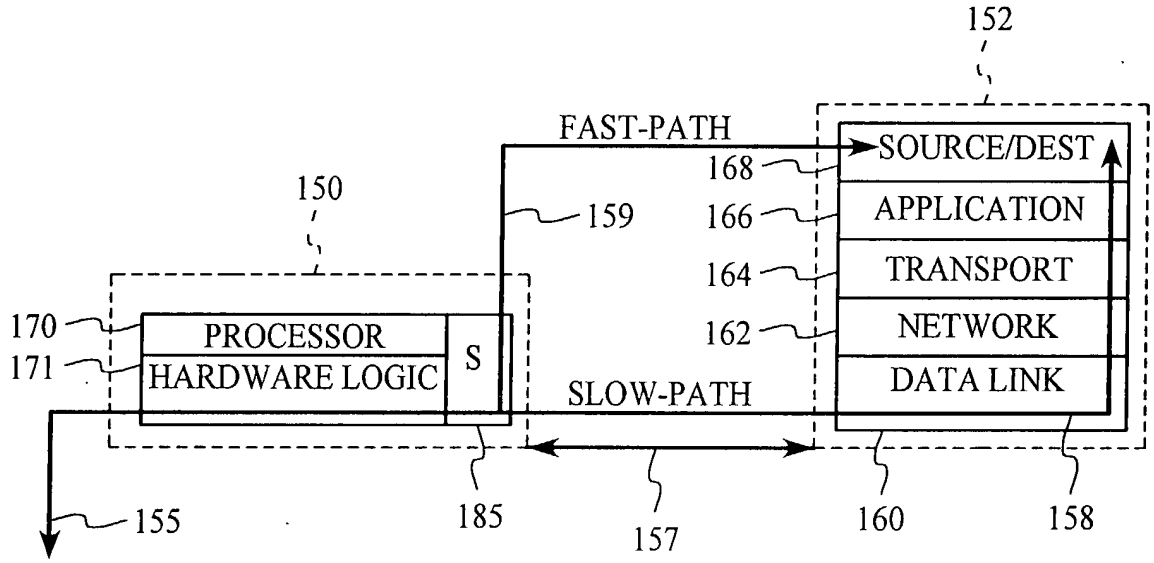

152
FAST-PATH
SOURCE/DEST
168
150
159
166
APPLICATION
164
TRANSPORT
170
PROCESSOR
S
162
NETWORK
171
HARDWARE LOGIC
SLOW-PATH
DATA LINK
155
185
157
160
158

FIG. 6

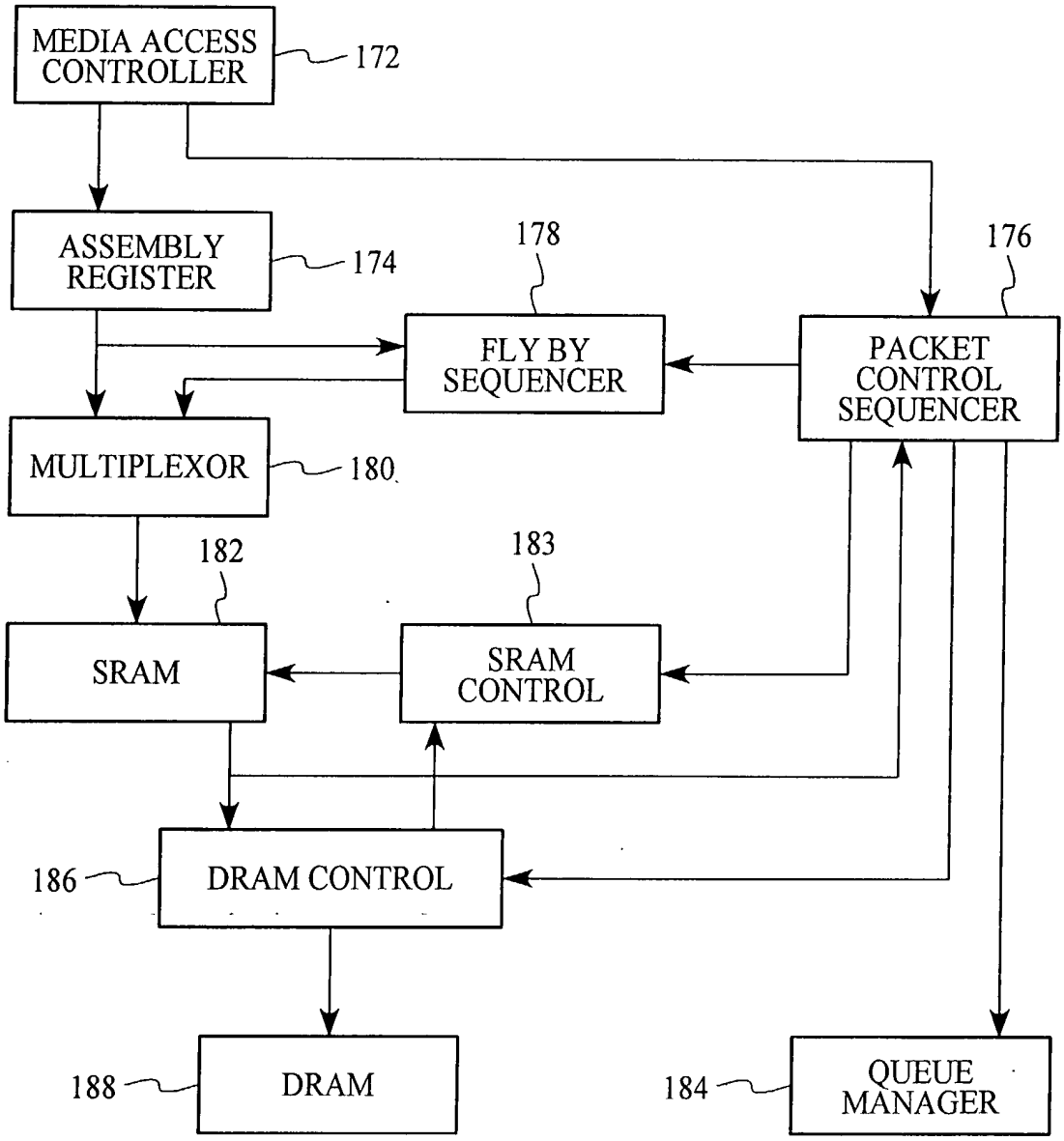

MEDIA ACCESS CONTROLLER
172
ASSEMBLY REGISTER
174
178
176
FLY BY SEQUENCER
PACKET CONTROL SEQUENCER
MULTIPLEXOR
180
182
183
SRAM
SRAM CONTROL
186
DRAM CONTROL
188
DRAM
184
QUEUE MANAGER

FIG. 7

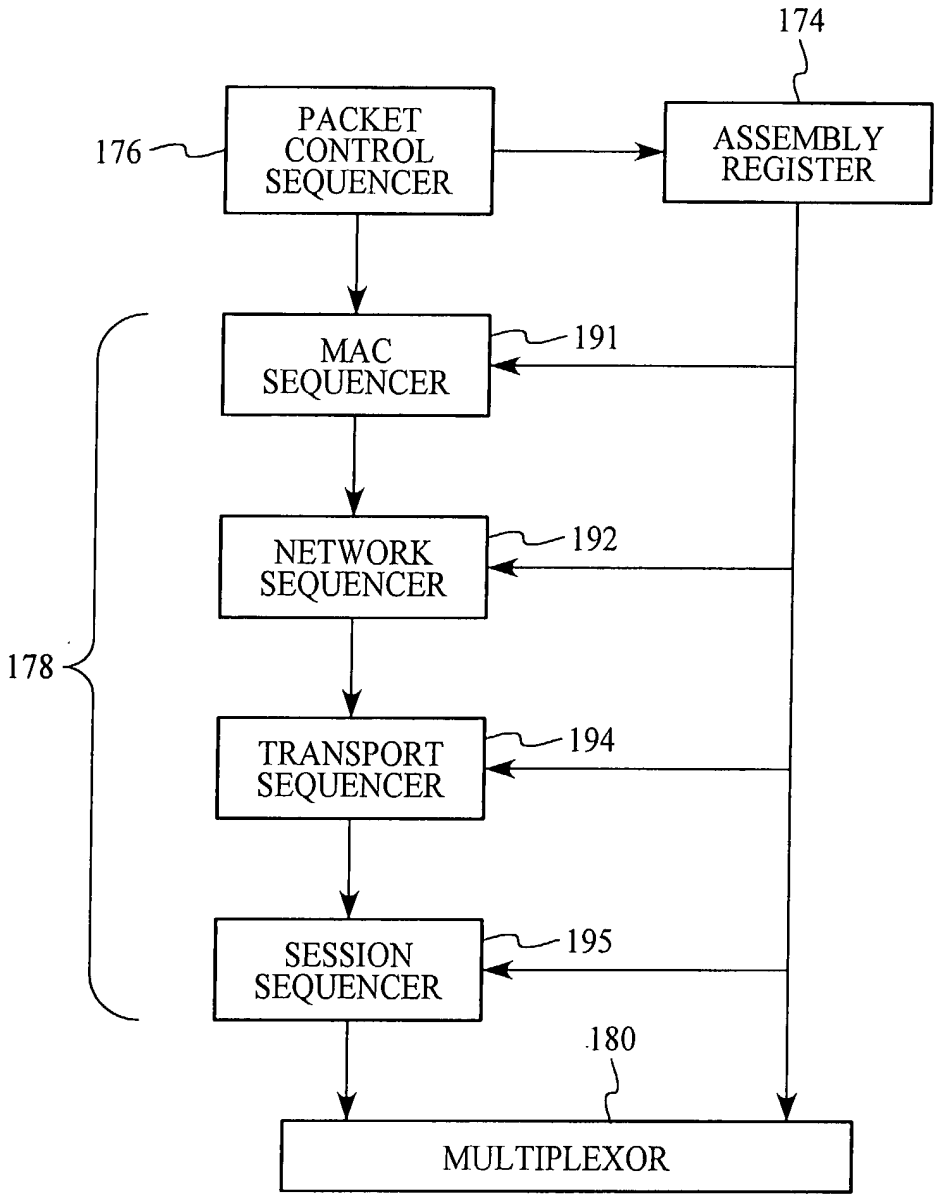

174
176
PACKET CONTROL SEQUENCER
ASSEMBLY REGISTER
MAC SEQUENCER
191
NETWORK SEQUENCER
192
178
TRANSPORT SEQUENCER
194
SESSION SEQUENCER
195
180
MULTIPLEXOR

FIG. 8

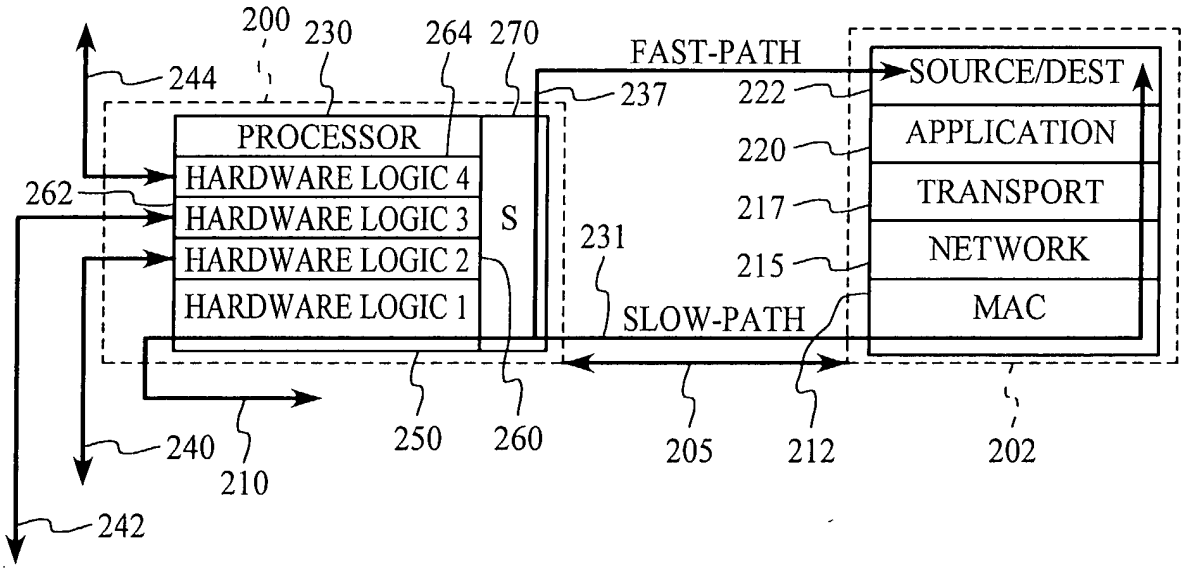

200
230
264
270
FAST-PATH
244
237
222
SOURCE/DEST
PROCESSOR
220
APPLICATION
262
HARDWARE LOGIC 4
HARDWARE LOGIC 3
S
217
TRANSPORT
231
HARDWARE LOGIC 2
215
NETWORK
HARDWARE LOGIC 1
SLOW-PATH
MAC
240
250
260
205
212
202
210
242

FIG. 9

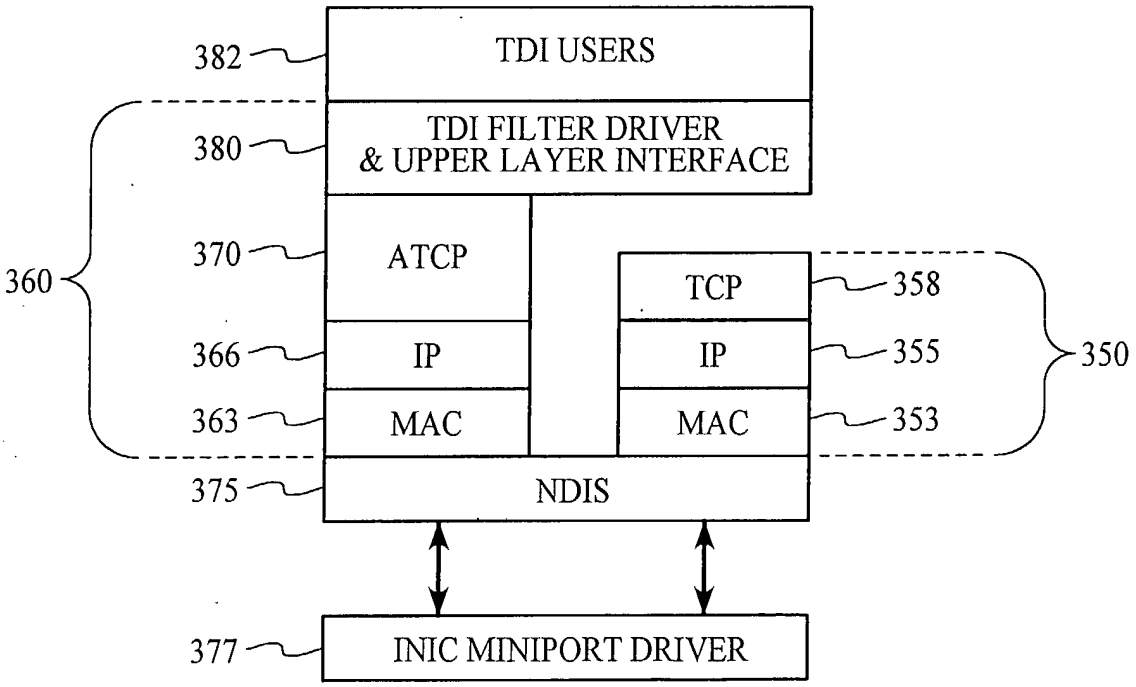

382
TDI USERS
380
TDI FILTER DRIVER
& UPPER LAYER INTERFACE
370
ATCP
360
TCP
358
366
IP
IP
355
350
363
MAC
MAC
353
375
NDIS
377
INIC MINIPORT DRIVER

FIG. 11

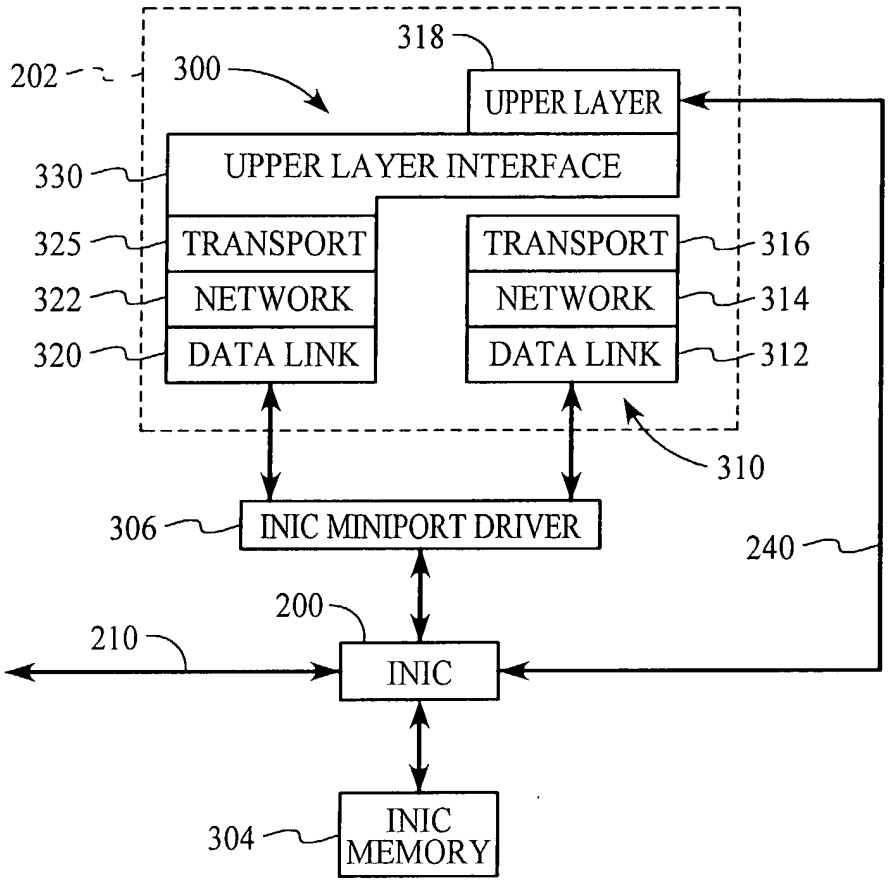

202
300
318
UPPER LAYER
330
UPPER LAYER INTERFACE
325
TRANSPORT
316
TRANSPORT
322
NETWORK
314
NETWORK
320
DATA LINK
312
DATA LINK
310
306
INIC MINIPORT DRIVER
240
200
210
INIC
304
INIC MEMORY

FIG. 10

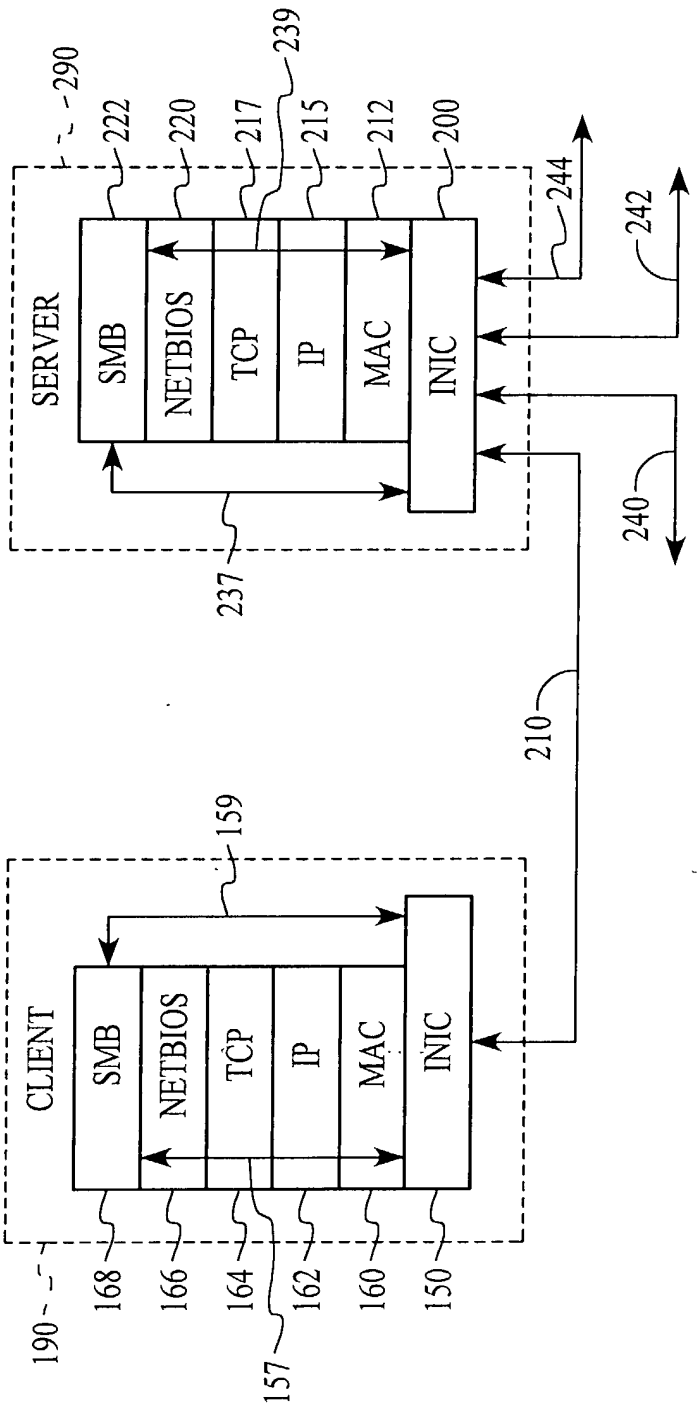

SERVER
SMB
NETBIOS
TCP
IP
MAC
INIC
290
222
220
217
239
215
212
200
244
242
240
237
210
159
CLIENT
SMB
NETBIOS
TCP
IP
MAC
INIC
190
168
166
164
157
162
160
150

FIG. 12

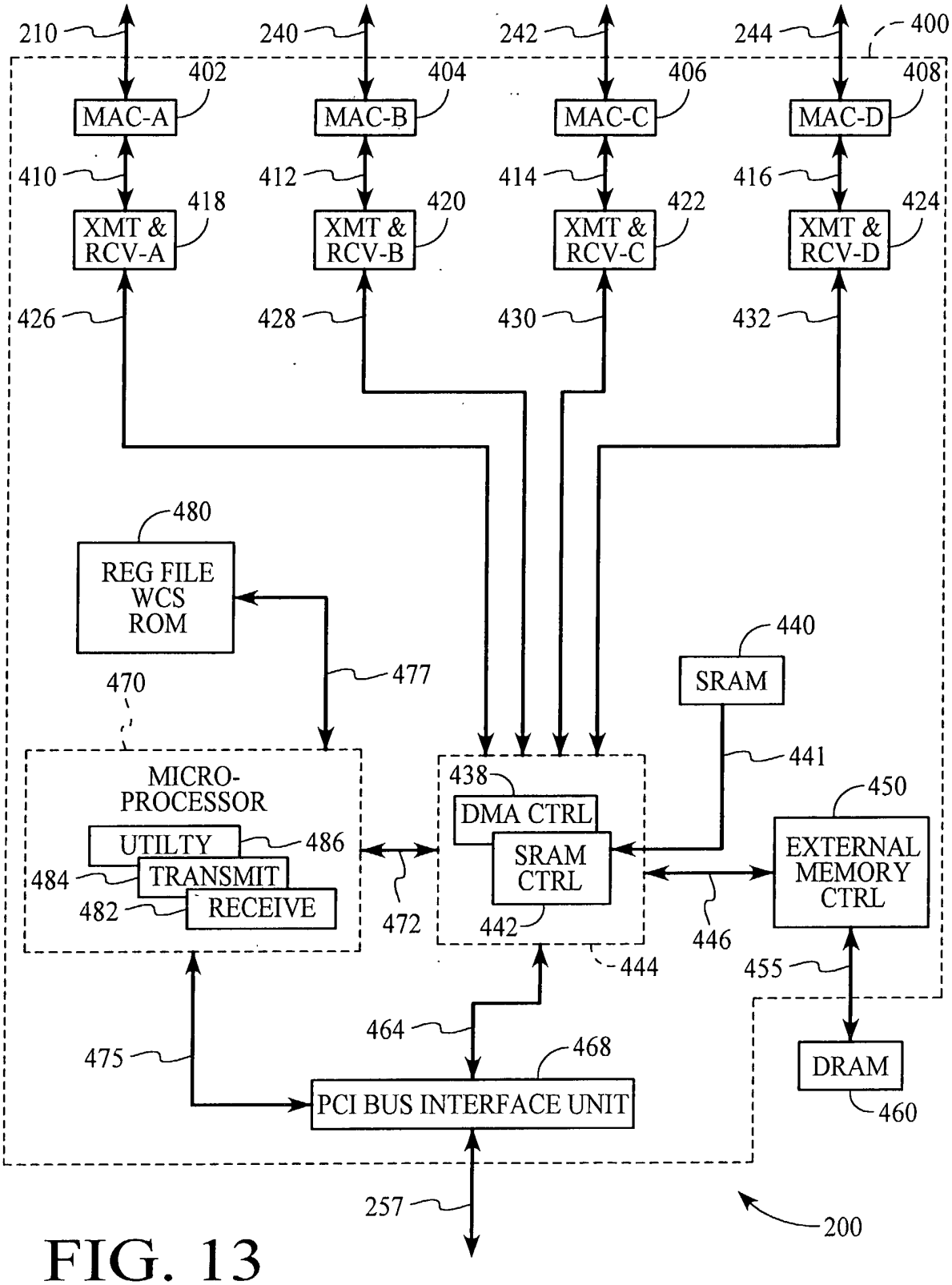

210
240
242
244
400
402
404
406
408
MAC-A
MAC-B
MAC-C
MAC-D
410
412
414
416
418
420
422
424
XMT & RCV-A
XMT & RCV-B
XMT & RCV-C
XMT & RCV-D
426
428
430
432
480
REG FILE WCS ROM
477
470
440
SRAM
441
MICRO-PROCESSOR
UTILTY
486
484
TRANSMIT
482
RECEIVE
472
438
DMA CTRL
SRAM CTRL
442
444
450
EXTERNAL MEMORY CTRL
446
455
464
468
475
PCI BUS INTERFACE UNIT
DRAM
460
257
200

FIG. 13

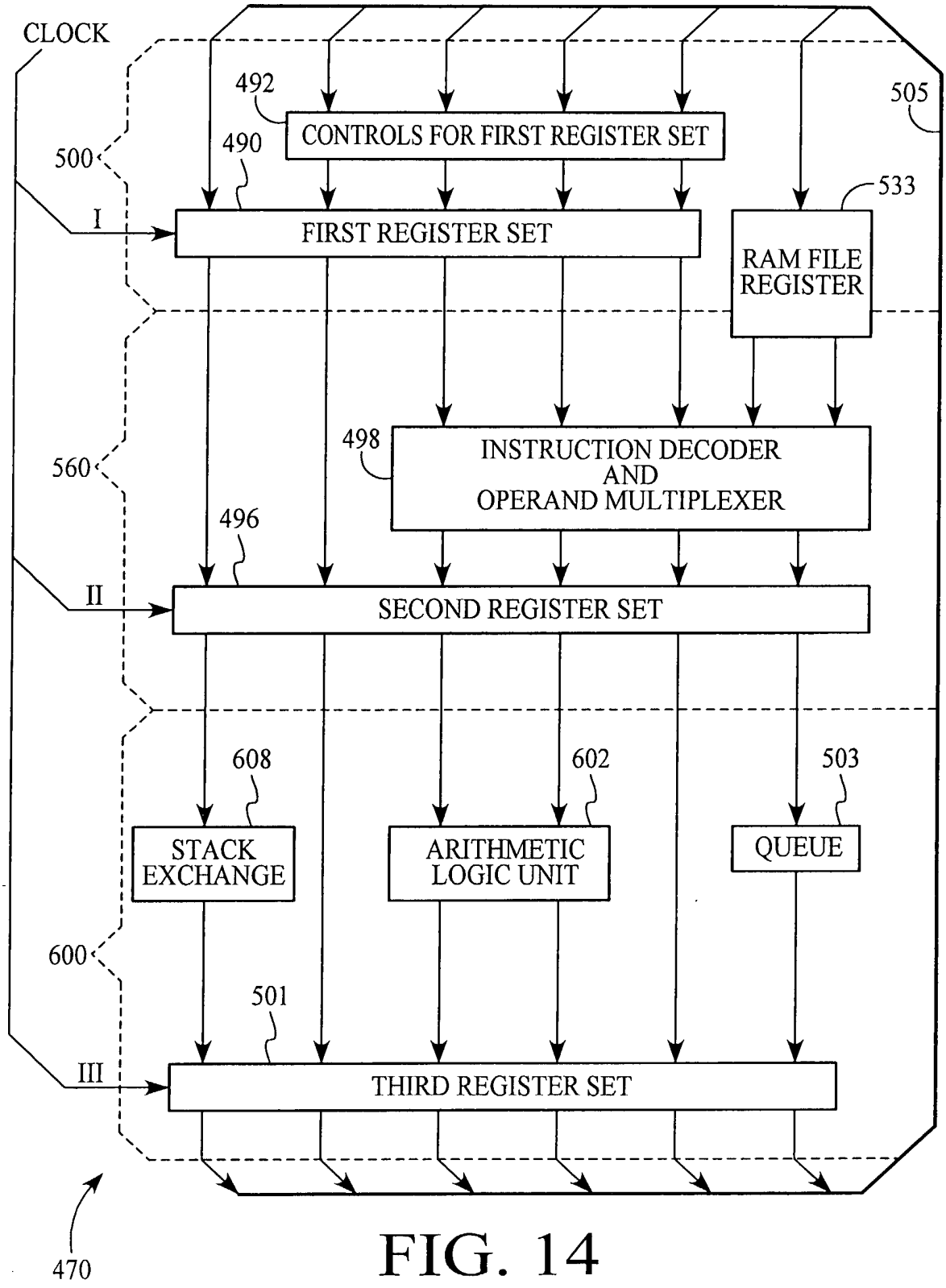

CLOCK
492
505
CONTROLS FOR FIRST REGISTER SET
500
490
533
I
FIRST REGISTER SET
RAM FILE REGISTER
498
INSTRUCTION DECODER AND OPERAND MULTIPLEXER
560
496
II
SECOND REGISTER SET
608
602
503
STACK EXCHANGE
ARITHMETIC LOGIC UNIT
QUEUE
600
501
III
THIRD REGISTER SET
470

FIG. 14

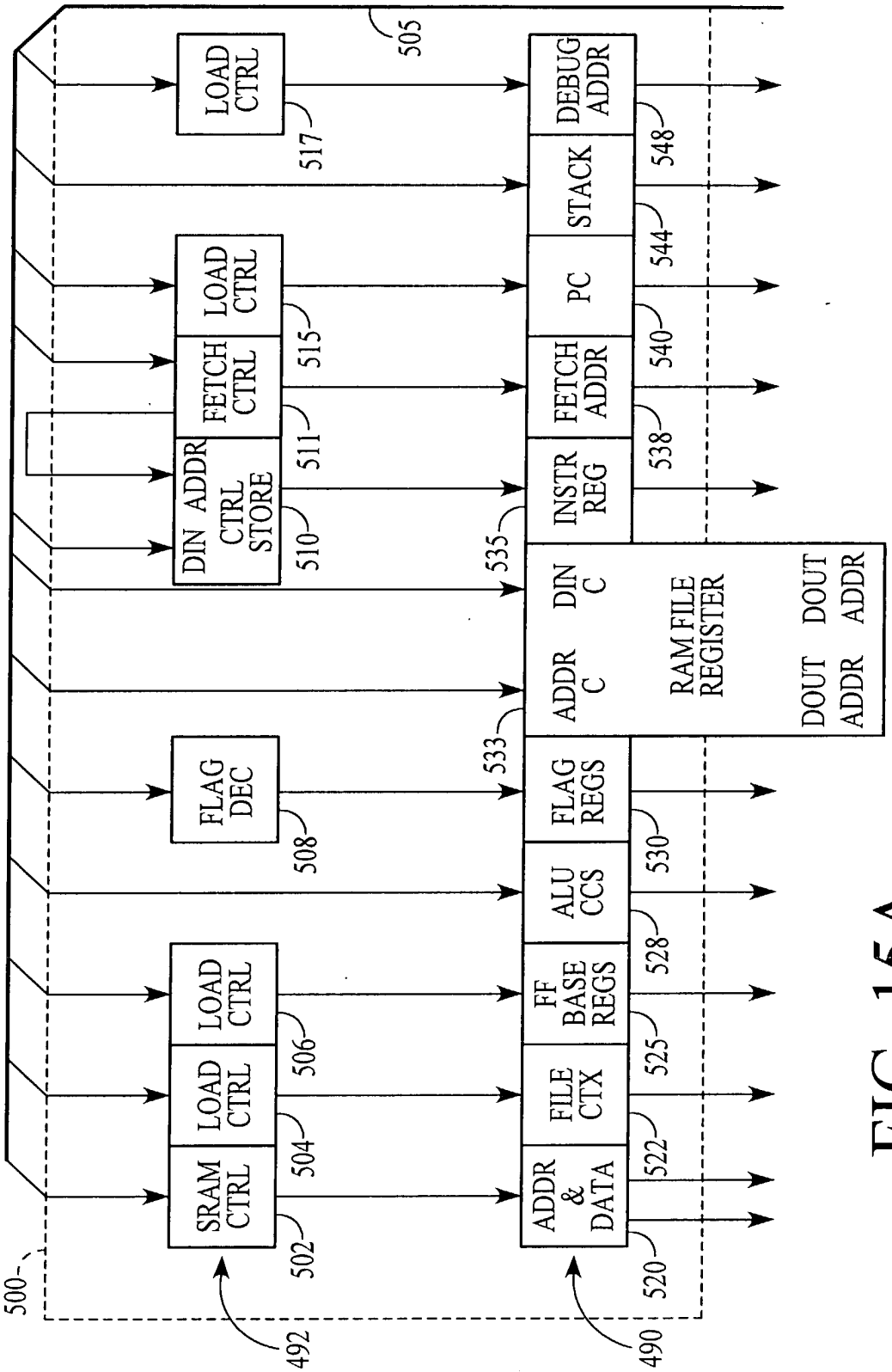

500
505
SRAM CTRL
LOAD CTRL
LOAD CTRL
LOAD CTRL
FLAG DEC
DIN ADDR CTRL STORE
FETCH CTRL
LOAD CTRL
ADDR & DATA
FILE CTX
FF BASE REGS
ALU CCS
FLAG REGS
ADDR C
DIN C
RAM FILE REGISTER
DOUT ADDR
DOUT ADDR
INSTR REG
FETCH ADDR
PC
STACK
DEBUG ADDR
LOAD CTRL
502
504
506
508
510
511
515
517
520
522
525
528
530
533
535
538
540
544
548
490
492

FIG. 15A

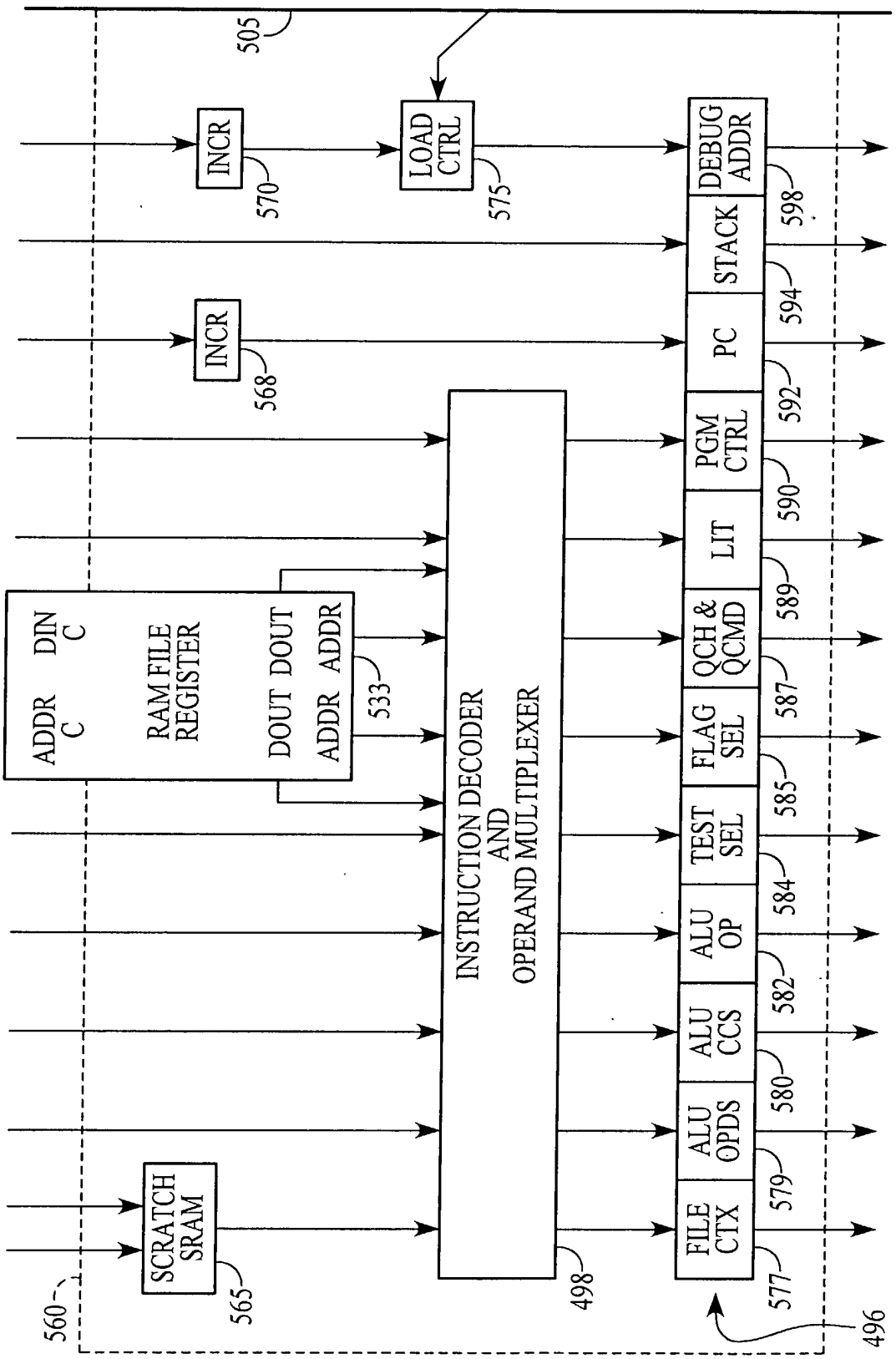

505
INCR
570
LOAD CTRL
575
DEBUG ADDR
598
STACK
594
INCR
568
PC
592
PGM CTRL
590
LIT
589
QCH & QCMD
587
FLAG SEL
585
TEST SEL
584
ALU OP
582
ALU CCS
580
ALU OPDS
579
FILE CTX
577
ADDR C
DIN C
RAM FILE REGISTER
DOUT DOUT
ADDR ADDR
533
INSTRUCTION DECODER AND OPERAND MULTIPLEXER
498
SCRATCH SRAM
565
560
496

FIG. 15B

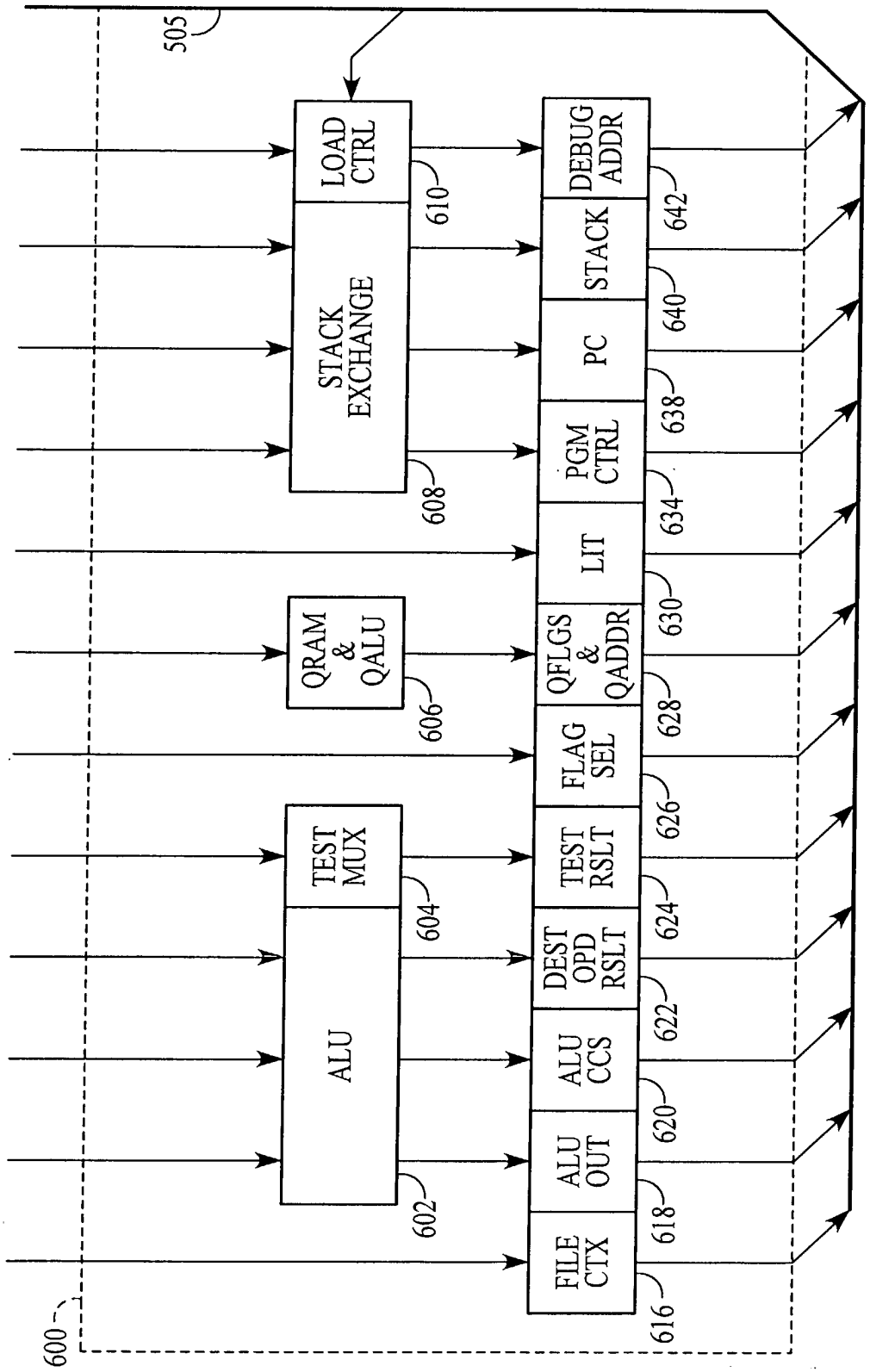

505
LOAD CTRL
STACK EXCHANGE
QRAM & QALU
TEST MUX
ALU
FILE CTX
ALU OUT
ALU CCS
DEST OPD RSLT
TEST RSLT
FLAG SEL
QFLGS & QADDR
LIT
PGM CTRL
PC
STACK
DEBUG ADDR
600
602
604
606
608
610
616
618
620
622
624
626
628
630
634
638
640
642

FIG. 15C

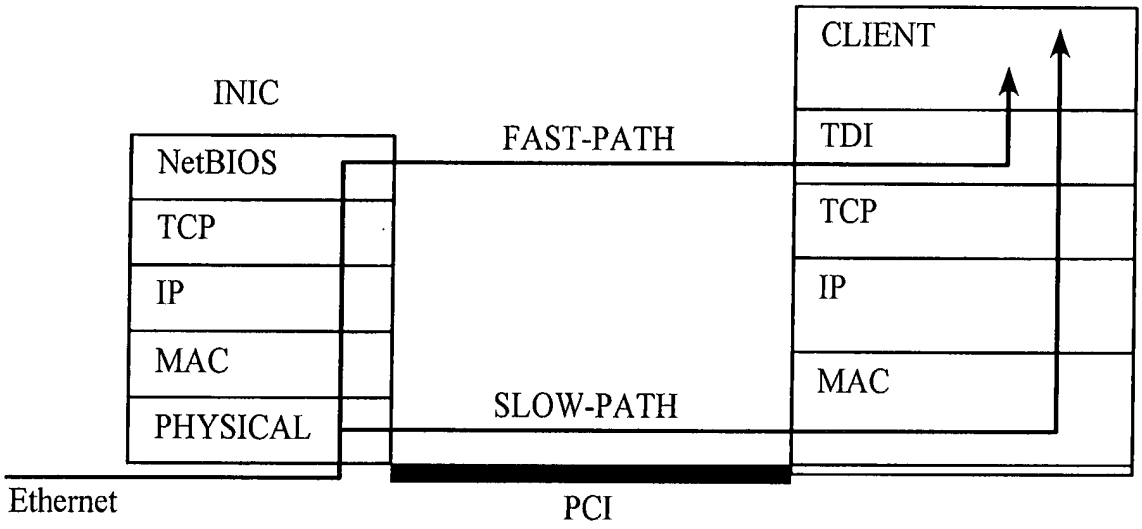

INIC
NetBIOS
TCP
IP
MAC
PHYSICAL
Ethernet
FAST-PATH
SLOW-PATH
PCI
CLIENT
TDI
TCP
IP
MAC

FIG. 16

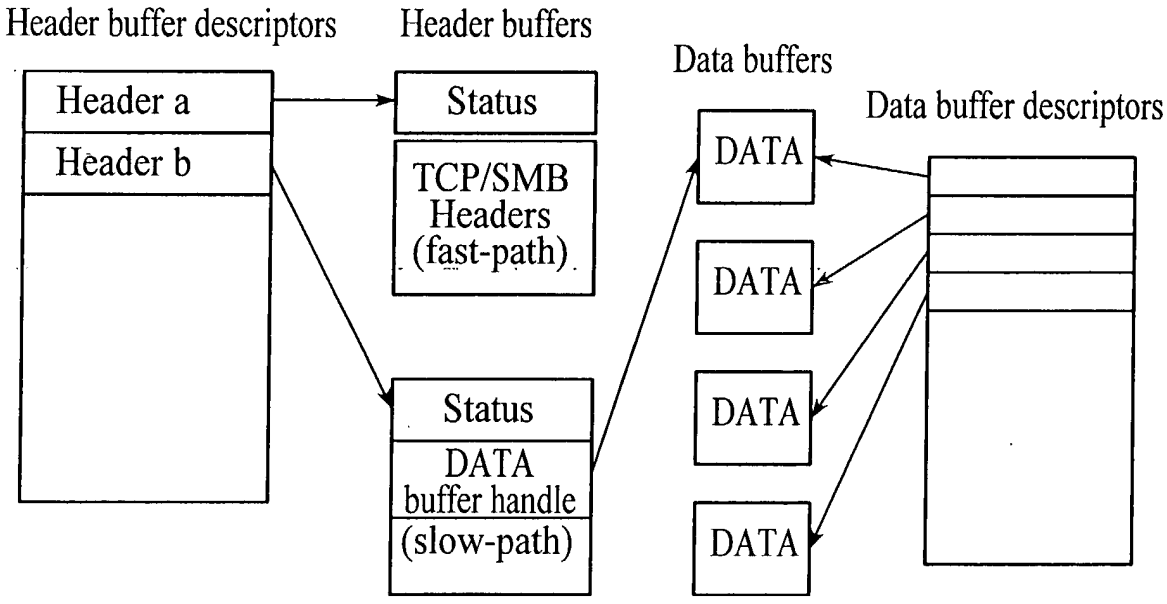

Header buffer descriptors
Header buffers
Data buffers
Data buffer descriptors
Header a
Header b
Status
TCP/SMB Headers (fast-path)
Status
DATA buffer handle (slow-path)
DATA
DATA
DATA
DATA

FIG. 17

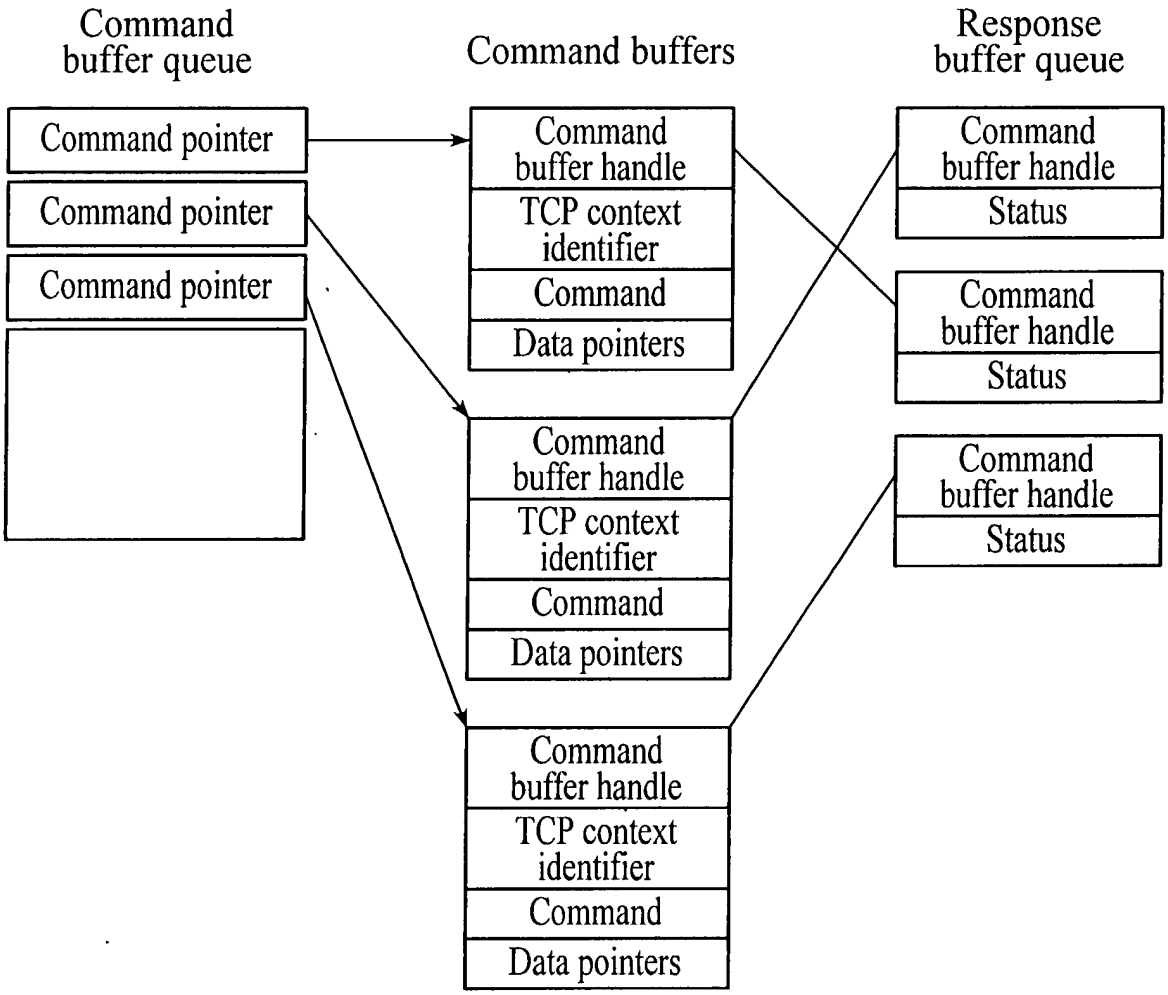

Command buffer queue
Command buffers
Response buffer queue
Command pointer
Command pointer
Command pointer
Command buffer handle
TCP context identifier
Command
Data pointers
Command buffer handle
Status
Command buffer handle
TCP context identifier
Command
Data pointers
Command buffer handle
Status
Command buffer handle
TCP context identifier
Command
Data pointers
Command buffer handle
Status

FIG. 18

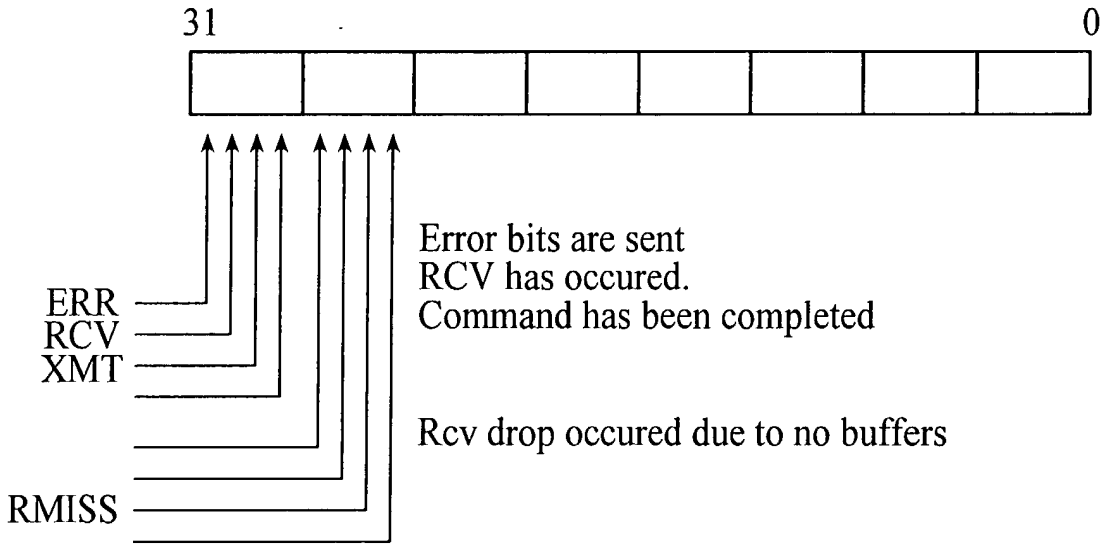


FIG. 19

ISR	0x0	Interrupt Status
IMR	0x4	Interrupt Mask
HBAR	0x8	Header Buffer Address
DBHR	0xC	Data Buffer Handle
DBAR	0x10	Data Buffer Address
CBAR0	0x14	Command Buffer Address XMT0
CBAR1	0x18	Command Buffer Address XMT1
CBAR2	0x1C	Command Buffer Address XMT2
CBAR3	0x20	Command Buffer Address XMT3
CBAR4	0x24	Command Buffer Address RCV
RBAR	0x28	Response Buffer Address

FIG. 20

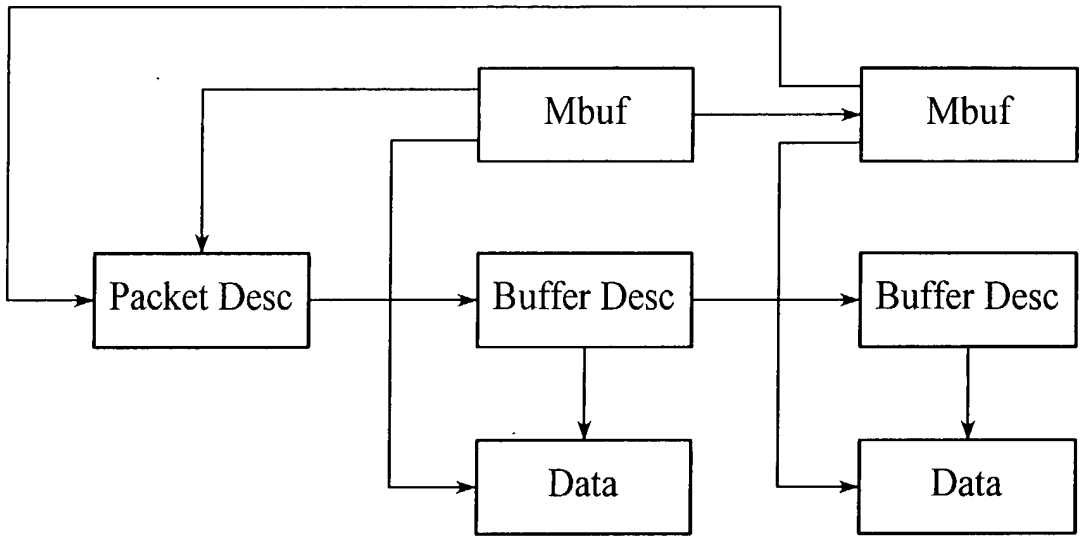

Mbuf
Mbuf
Packet Desc
Buffer Desc
Buffer Desc
Data
Data

FIG. 21

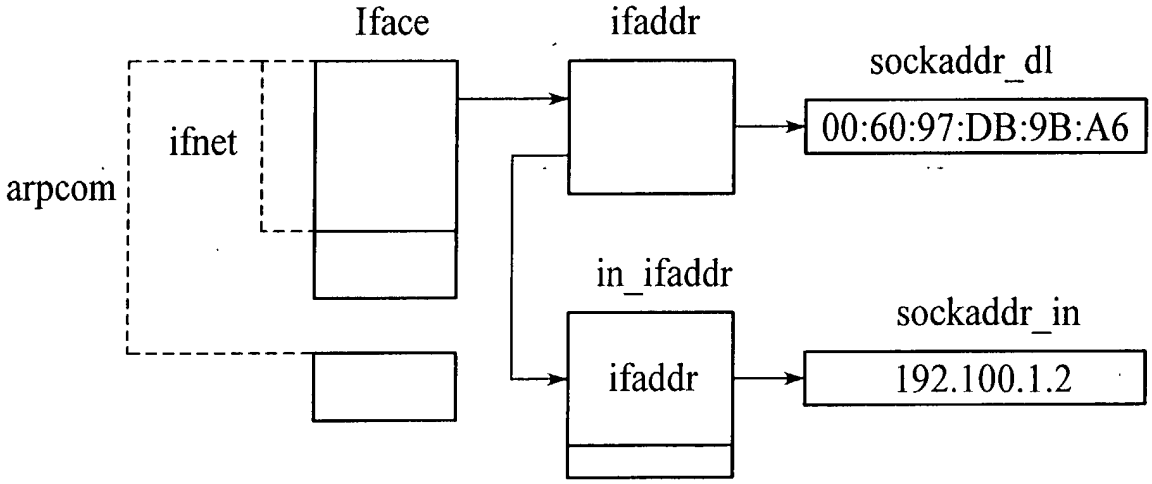

Iface
ifaddr
sockaddr_dl
ifnet
00:60:97:DB:9B:A6
arpcom
in_ifaddr
sockaddr_in
ifaddr
192.100.1.2

FIG. 22

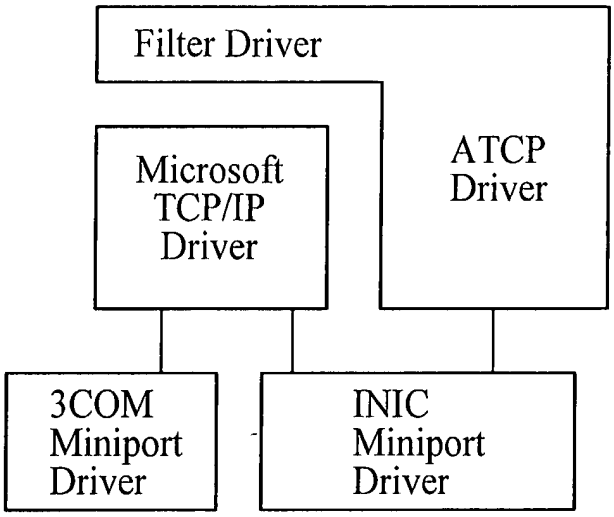
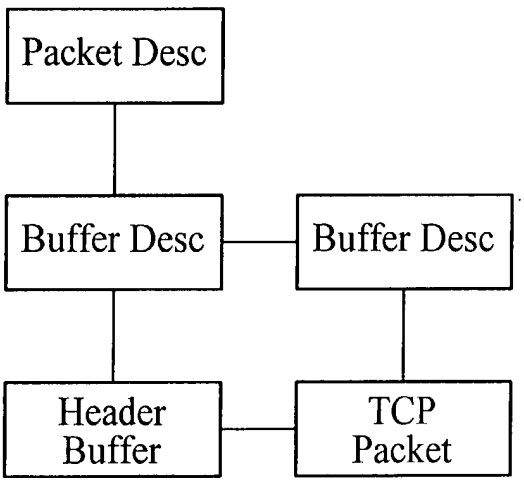
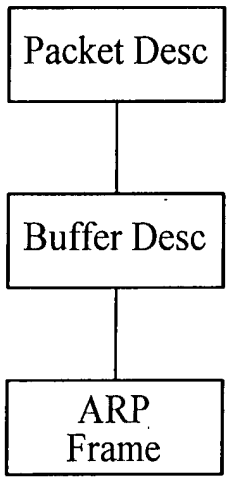


FIG. 23



Example of incoming TCP pkt

FIG. 24



Example of incoming ARP Frame

FIG. 25

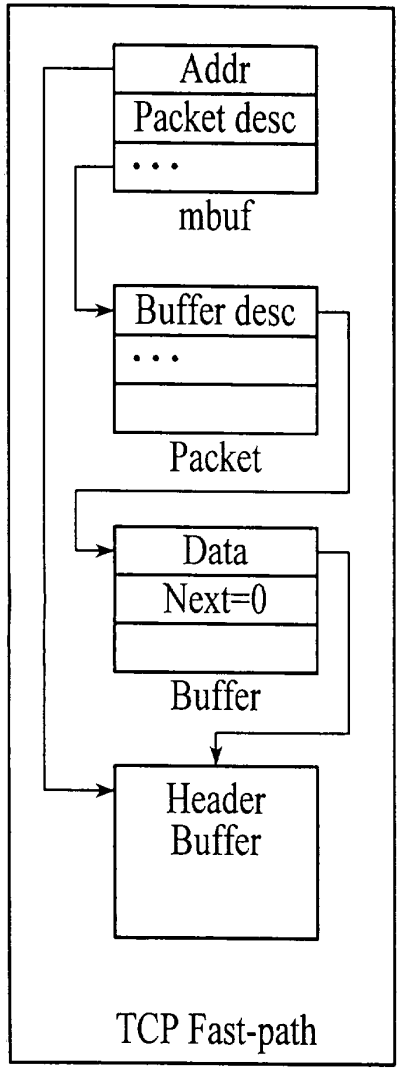


FIG. 26A

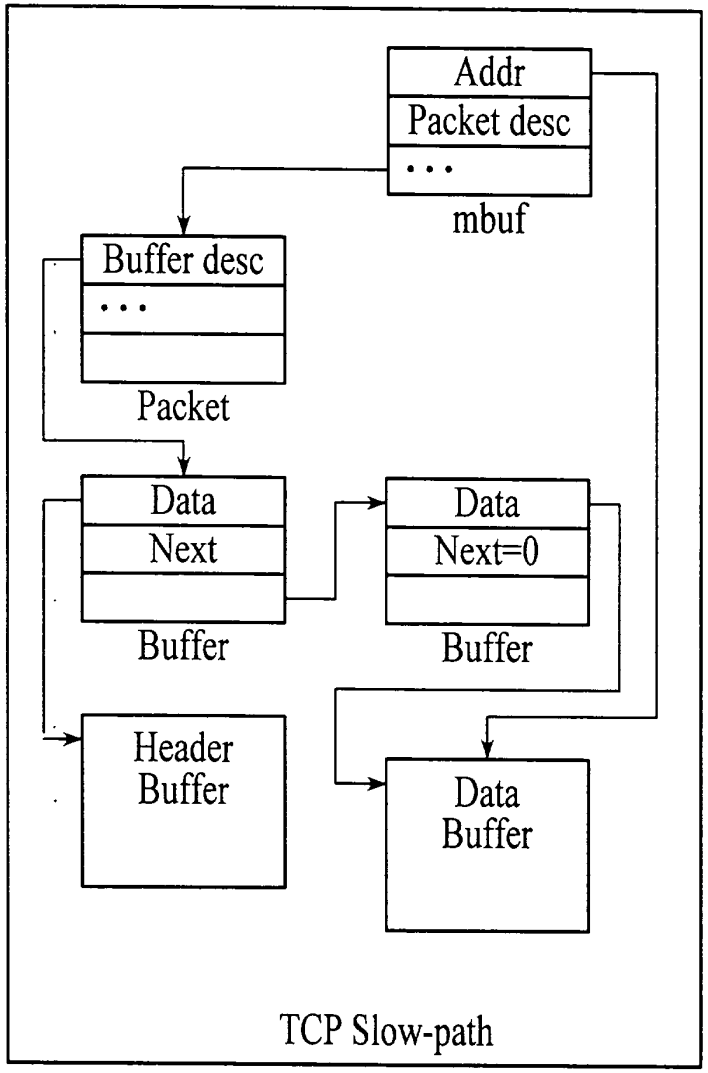


FIG. 26B

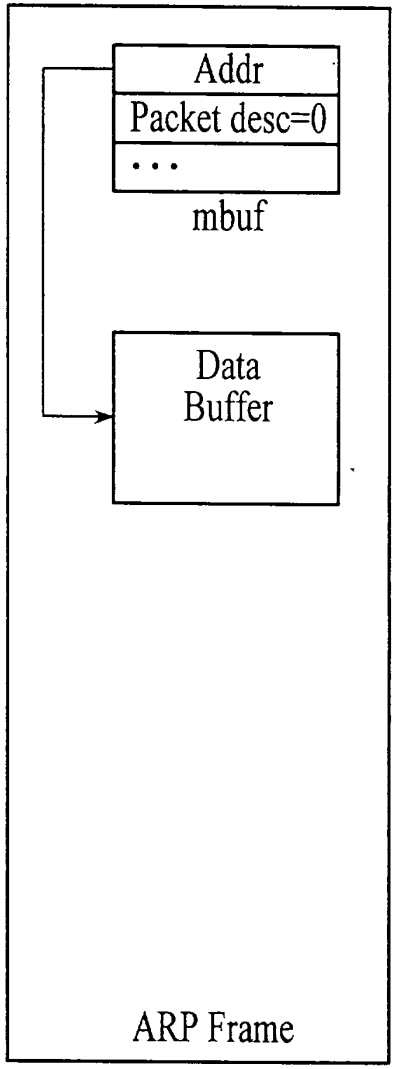


FIG. 26C

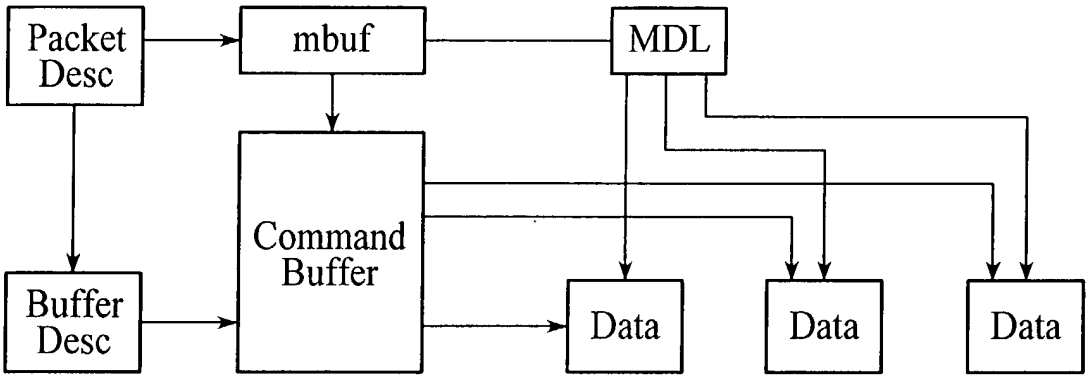

Packet Desc
mbuf
MDL
Command Buffer
Buffer Desc
Data
Data
Data

FIG. 27

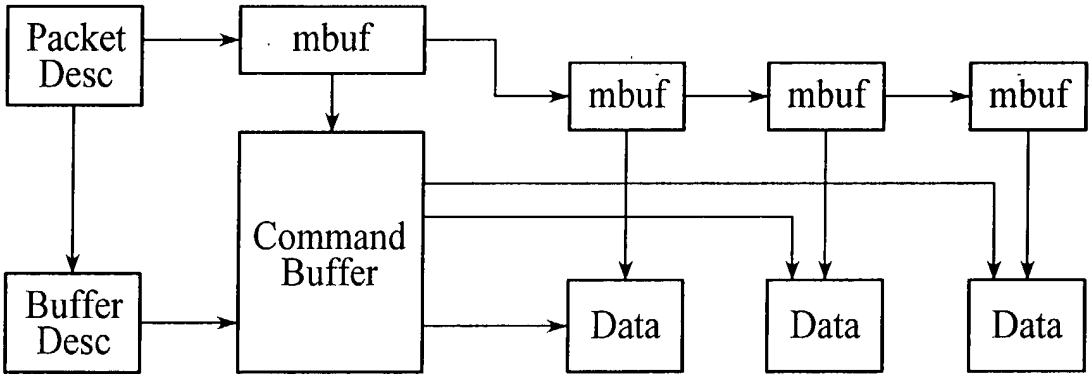

Packet Desc
mbuf
mbuf
mbuf
mbuf
Command Buffer
Buffer Desc
Data
Data
Data

FIG. 28

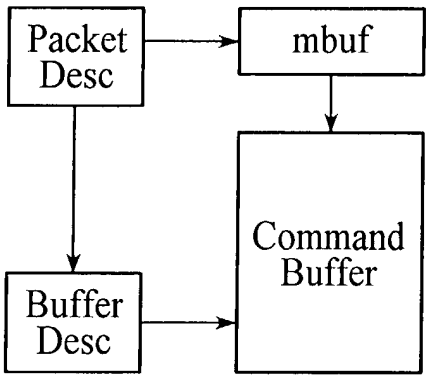


FIG. 29

SRAM requirements for the Receive and Transmit engines:

TCB buffers	256 bytes* 16	4096
Header buffers	128 bytes* 16	2048
TCB hash index	16 bytes* 256	4096
Timers		128
DRAM Fifo queues	128 bytes* 16	2048
		~12K bytes

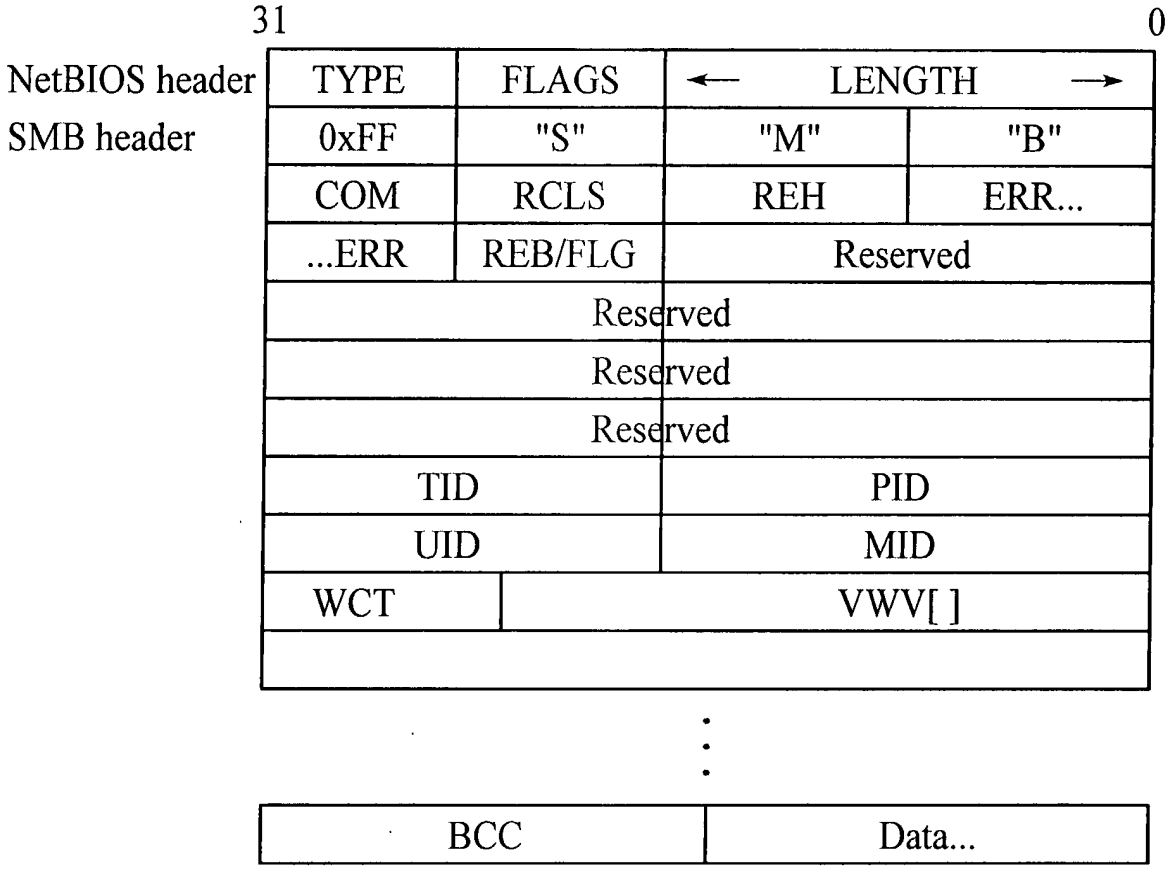
FIG. 30

Summary of the main loop of Receive:

```
forever {
        while there are any Receive events {
                if (a new event) {
                        if (no new context available)
                                ignore the event;
                }
                call appropriate event handler to service the event;
                this may make a waiting process runnable or set up
                a new process to be run (get free context, hddr buffer,
                TCB buffer, set the context up).
        }
        while any process contexts are runable {
                run them by jumping to the start/resume address;
                if (process complete)
                        free the context;
        }
}
```

FIG. 31

Format of the SMB header of an SMB frame:



Notes (interesting fields):

LENGTH	17 bit Length of SMB message (0 - 128K)
COM	SMB command
WCT	Count (16 bit) of parameter words in VWV [ ]
VWV	Variable number of parameter words
BCC	Bytes of data following

FIG. 32

Summary of the main loop of Transmit:

```
forever {
        while there are any Transmit events {
                if (a new event) {
                        if (no new context available)
                                ignore the event;
                }
                call appropriate event handler to service the event;
                this may make a waiting process runnable or set up
                a new process to be run (get free context, hddr buffer,
                TCB buffer, set the context up).
        }
        while any process contexts are runable {
                run them by jumping to the start/resume address;
                if (process complete)
                        free the context;
        }
}
```

FIG. 33

Bit 31 - 24 Byte enable 7 - 0. Only the low order four bits are valid for 32 bit addressing mode.
Bit 23 - 0 Memory access
1 Configuration access
Bit 22 - 0 Read (to Host)
1 Write (to Host)
Bit 21 - 1 Data Valid
Bit 20 - 16 Reserved
Bit 15 - 0 Address

FIG. 34

Configuration Space 1	SRAM Address Offset
00	00
04	04
08	08
0C	0C
10	10
3C	14

Configuration Space 2

00	00
04	18
08	08
0C	1C
10	20
3C	24

All other reads to configuration space will return 00.

FIG. 35

Bit 0 - 0 I/O accesses are not enabled
Bit 1 - 1 Memory accesses are enabled
Bit 2 - 1 Bus master is enabled
Bit 3 - 0 Special Cycle is not enabled
Bit 4 - 1 Memory Write and Invalidate is enabled
Bit 5 - 0 VGA palette snooping is not enabled
Bit 6 - 1 Parity checking is enabled
Bit 7 - 0 Address data stepping is not enabled
Bit 8 - SERR# is enabled
Bit 9 - 0 Fast back to back is not enabled

FIG. 36

Bit 5 - 1 66 MHz capable is enabled. This bit will be set if the INIC Detects the system running at 66 MHz on reset
Bit 6 - 0 User Definable Features is not enabled
Bit 7 - 1 Fast Back-to-Back slave transfers enabled
Bit 8 - 1 Parity Error enabled - This bit is initialized to 0
Bit 9,10 - 00 - Fast device select will be set if we are at 33 MHz
01 - Medium device select will be set if we are at 66 MHz
Bit 11 - 1 Target Abort is implemented. Initialized to 0.
Bit 12 - 1 Target Abort is implemented. Initialized to 0.
Bit 13 - 1 Master Abort is implemented. Initialized to 0.
Bit 14 - 1 SERR# is implemented. Initialized to 0.
Bit 15 - 1 Parity error is implemented. Initialized to 0.

FIG. 37

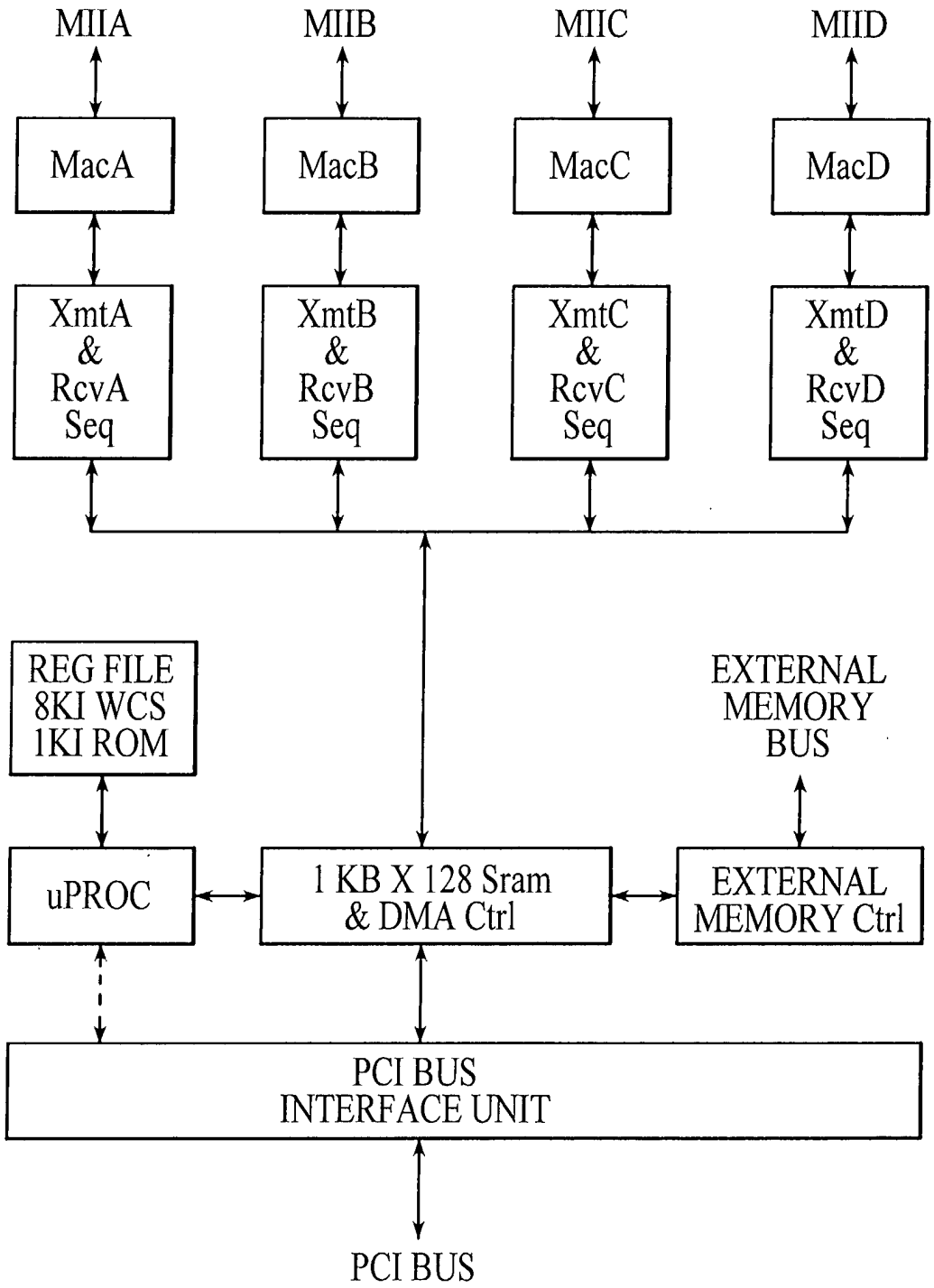

MIIA
MIIB
MIIC
MIID
MacA
MacB
MacC
MacD
XmtA & RcvA Seq
XmtB & RcvB Seq
XmtC & RcvC Seq
XmtD & RcvD Seq
REG FILE 8KI WCS 1KI ROM
EXTERNAL MEMORY BUS
uPROC
1 KB X 128 Sram & DMA Ctrl
EXTERNAL MEMORY Ctrl
PCI BUS INTERFACE UNIT
PCI BUS

FIG. 38

MODULE	DESCR	SPEED	AREA
Scratch RAM,	1Kx128 sport,	4.37 ns nom.,	06.77 mm^2
WCS,	8Kx49 sport,	6.40 ns nom.,	18.29 mm^2
MAP,	128x7 sport,	3.50 ns nom.,	00.24 mm^2
ROM,	1Kx49 32col,	5.00 ns nom.,	00.45 mm^2
REGs,	512x32 tport,	6.10 ns nom.,	03.49 mm^2
Macs,	.75 mm^2 x 4 =		03.30 mm^2
PLL,	.5 mm^2 =		00.55 mm^2
MISC LOGIC,	117,260 gates / (5035 gates / $mm^{2)}$ =		23.29 mm^2
TOTAL CORE			56.22 mm^2

$(\text{Core side})^2$		=	56.22 mm^2
Core side		=	07.50 mm
Die side	= core side + 1.0 mm (I/O cells)	=	08.50 mm
Die area	= 8.5 mm x 8.5 mm	=	72.25 mm^2
Pads needed	= 220 signals x 1.25 (vss, vdd)	=	275 pins
LSI PBGA		=	272 pins

FIG. 39

(10MB/s/100Base) x 2 (full duplex) x 4 connections	=	80 MB/s
Average frame size	=	512 B
Frame rate = 80MB/s / 512B	=	156,250 frames / s
Cpu overhead / frame = (256B context read) + (64B header read) + (128B context write) + (128B misc.)	=	512B / frame
Total bandwidth = (512B in) + (512B out) + (512B Cpu)	=	1536B / frame
Dram Bandwidth required = (1536B/frame) x (156,250 frames/s)	=	240MB/s
Dram Bandwidth @ 60MHz = (32 bytes / 167ns)	=	202MB/s
Dram Bandwidth @ 66MHz = (32 bytes / 150ns)	=	224MB/s
PCI Bandwidth required	=	80MB/s
PCI Bandwidth available @ 30 MHz, 32b, average	=	46MB/s
PCI Bandwidth available @ 33 MHz, 32b, average	=	50MB/s
PCI Bandwidth available @ 60 MHz, 32b, average	=	92MB/s
PCI Bandwidth available @ 66 MHz, 32b, average	=	100MB/s
PCI Bandwidth available @ 30 MHz, 64b, average	=	92MB/s
PCI Bandwidth available @ 33 MHz, 64b, average	=	100MB/s
PCI Bandwidth available @ 60 MHz, 64b, average	=	184MB/s
PCI Bandwidth available @ 66 MHz, 64b, average	=	200MB/s

FIG. 40

Receive frame interval = 512B / 40MB/s	=	12.8us
Instructions / frame @ 60MHz = (12.8us/frame) / (50ns/instruction) instructions/frame	=	256
Instructions / frame @ 66MHz = (12.8us/frame) / (45ns/instruction) instructions/frame	=	284
Required instructions / frame	=	250 instructions/frame

FIG. 41

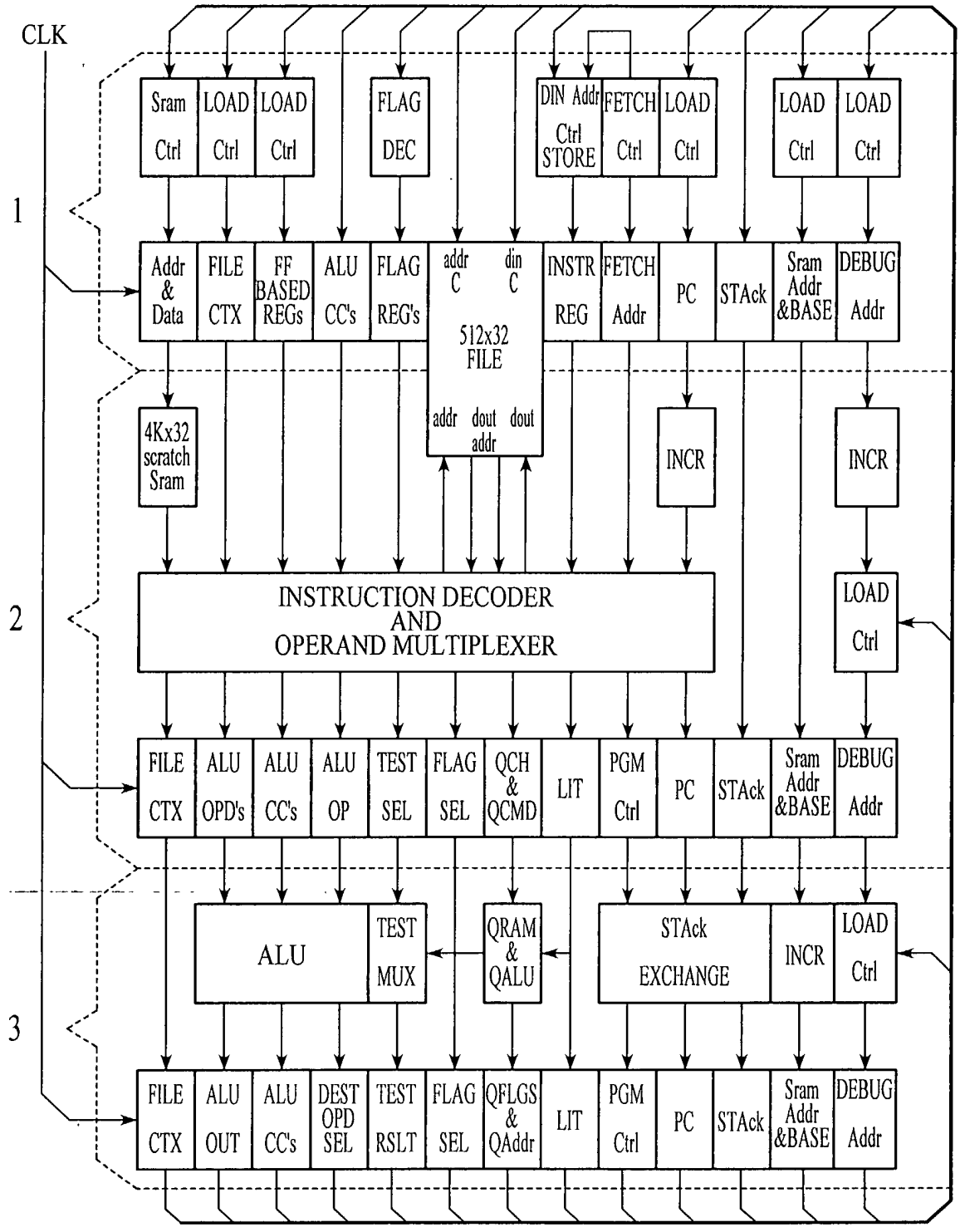

CLK
1
2
3
Sram Ctrl
LOAD Ctrl
LOAD Ctrl
FLAG DEC
DIN Addr Ctrl STORE
FETCH Ctrl
LOAD Ctrl
LOAD Ctrl
LOAD Ctrl
Addr & Data
FILE CTX
FF BASED REGs
ALU CC's
FLAG REG's
addr C
din C
512x32 FILE
addr dout addr
dout addr
INSTR REG
FETCH Addr
PC
STAck
Sram Addr &BASE
DEBUG Addr
4Kx32 scratch Sram
INCR
INCR
INSTRUCTION DECODER AND OPERAND MULTIPLEXER
LOAD Ctrl
FILE CTX
ALU OPD's
ALU CC's
ALU OP
TEST SEL
FLAG SEL
QCH & QCMD
LIT
PGM Ctrl
PC
STAck
Sram Addr &BASE
DEBUG Addr
ALU
TEST MUX
QRAM & QALU
STAck EXCHANGE
INCR
LOAD Ctrl
FILE CTX
ALU OUT
ALU CC's
DEST OPD SEL
TEST RSLT
FLAG SEL
QFLGS & QAddr
LIT
PGM Ctrl
PC
STAck
Sram Addr &BASE
DEBUG Addr

FIG. 42

INSTRUCTION-WORD FORMAT

TYPE	[55:49]	[48:47]	[46:42]	[41:33]	[32:24]	[23:16]	[15:00]
Jcc	0b0000000	0b00,	**AluOp,**	**OpdASel,**	**OpdBSel,**	**TstSel,**	**Literal**
Jmp	0b0000000	0b01,	**AluOp,**	**OpdASel,**	**OpdBSel,**	**FlgSel,**	**Literal**
Jsr	0b0000000	0b10,	**AluOp,**	**OpdASel,**	**OpdBSel,**	**FlgSel,**	**Literal**
Rts	0b0000000	0b11,	**AluOp,**	**OpdASel,**	**OpdBSel,**	0hff,	**Literal**
Nxt	0b0000000	0b11,	**AluOp,**	**OpdASel,**	**OpdBSel,**	**FlgSel,**	**Literal**
Map	**MapAddr**	0BXX,	0BXXXXX,	0BXXXXXXXXX,	0BXXXXXXXXX,	0HXX,	0HXXXX

FIG. 43

SEQUENCER BEHAVIOR

```
if (MapEn & (MapAddr != 0b0000000)){                    //re-map instr
        Stackc = Stackc;
        StackB = StackB;
        StackA = StackA;
     InstrAddr = 0h8000 | Pc[2:0] | (MapAddr << 3);
         Pc = InstrAddr + (Execute & ~DbgMd);
      Fetch = DbgMd ? DbgAddr:InstrAddr;
    DbgAddr = DbgAddr + (Execute & DbgMd);}

else if (PgmCtrl == Jcc){                               //conditional jump
        Stackc = Stackc;
        StackB = StackB;
        StackA = StackA;
     InstrAddr = ~Tst@TstSel ? Pc:(AluDst==Pc) ? AluOut:Literal;
         Pc = InstrAddr + (Execute & ~DbgMd)
      Fetch = DbgMd ? DbgAddr:InstrAddr;
    DbgAddr = DbgAddr + (Execute & DbgMd);}

else if (PgmCtrl == Jmp){                               //jump
        Stackc = Stackc;
        StackB = StackB;
        StackA = StackA;
     InstrAddr = (AluDst == Pc) ? AluOut:Literal;
         Pc = InstrAddr + (Execute & ~DbgMd)
      Fetch = DbgMd ? DbgAddr:InstrAddr;
    DbgAddr = DbgAddr + (Execute & DbgMd);}

else if (PgmCtrl == Jsr){                               //jump subroutine
        Stackc = StackB;
        StackB = StackA;
        StackA = Pc;
     InstrAddr = (AluDst == Pc) ? AluOut:Literal;
         Pc = InstrAddr + (Execute & ~DbgMd)
      Fetch = DbgMd ? DbgAddr:InstrAddr;
    DbgAddr = DbgAddr + (Execute & DbgMd);}

else if (FlgSel == Rts){                                //return subroutine
     InstrAddr = StackA;
        StackA = StackB;
        StackB = Stackc;
        Stackc = ErrVec;
         Pc = InstrAddr + (Execute & ~DbgMd)
      Fetch = DbgMd ? DbgAddr:InstrAddr;
    DbgAddr = DbgAddr + (Execute & DbgMd);}

else    {
     InstrAddr = Pc;                                    //continue
        StackA = StackA;
        StackB = StackB;
        Stackc = Stackc;
         Pc = InstrAddr + (Execute & ~DbgMd)
      Fetch = DbgMd ? DbgAddr:InstrAddr;
    DbgAddr = DbgAddr + (Execute & DbgMd);}
```

FIG. 44

ALU OPERATIONS

AluOp	OPERATION	
0b00000	**A** = (**A** & ~(1 << **B**)); **C** = 0; **V** = (**B** >= 32) ? 1:0;	//bit clear
0b00001	**A** = (**A** & **B**); **C** = 0; **V** = 0;	//logical and
0b00010	**A** = (**Literal** & **B**); **C** = 0; **V** = 0;	//logical and
0b00011	**A** = (~**Literal** & **B**); **C** = 0; **V** = 0;	//logical and not
0b00100	**A** = (**A** | (1 << **B**)); **C** = 0; **V** = (**B** >= 32) ? 1:0;	//bit set
0b00101	**A** = (**A** | **B**); **C** = 0; **V** = 0;	//logical or
0b00110	**A** = (**Literal** | **B**); **C** = 0; **V** = 0;	//logical or
0b00111	**A** = (~**Literal** | **B**); **C** = 0; **V** = 0;	//logical or not
0b01000	for (i=31; i>=0; i--) if **B**[i] continue; **A**=i; **C** = 0; **V** = (**B**) ? 0:1;	//priority enc
0b01001	**A** = (**A** ^ **B**); **C** = 0; **V** = 0;	//logical xor
0b01010	**A** = ({**Literal**} ^ **B**); **C** = 0; **V** = 0;	//logical xor
0b01011	**A** = ({~**Literal**} ^ **B**); **C** = 0; **V** = 0;	//logical xor not
0b01100	**A** = **B**; **C** = 0; **V** = 0;	//move
0b01101	**A** = **B**[31:24] ^ **B**[23:16] ^ **B**[15:08] ^ **B**[07:00]; **C** = 0; **V** = 0;	//hash
0b01110	**A** = {**B**[23:16],**B**[31:24],**B**[07:00],**B**[15:08]}; **C** = 0; **V** = 0;	//swap bytes
0b01111	**A** = {**B**[15:00], **B**[31:16]}; **C** = 0; **V** = 0;	//swap doublets

FIG. 45

AluOp	FUNCTION	
0b10000	**A** = (**A** + **B**); **C** = (**A** + **B**)[32]; **V** = 0;	//add **B**
0b10001	**A** = (**A** + **B** + **C**); **C** = (**A** + **B** + **C**)[32]; **V** = 0;	//add **B, carry**
0b10010	**A** = (**Literal** + **B**); **C** = (**Literal** + **B**)[32]; **V** = 0;	//add constant
0b10011	**A** = (-**Literal** + **B**); **C** = (-**Literal** + **B**)[32]; **V** = 0;	//sub constant
0b10100	**A** = (**A** - **B**); **C** = (**A** - **B**)[32]; **V** = 0;	//sub **B**
0b10101	**A** = (**A** - **B** - ~**C**); **C** = (**A** - **B** - ~**C**)[32]; **V** = 0;	//sub **B, borrow**
0b10110	**A** = (-**A** + **B**); **C** = (-**A** + **B**)[32]; **V** = 0;	//sub **A**
0b10111	**A** = (-**A** + **B** - ~**C**); **C** = (-**A** + **B** - ~**C**)[32]; **V** = 0;	//sub **A, borrow**
0b11000	**A** = (**A** << **B**); **C** = **A**[31]; **V** = (**B** >= 32) ? 0:1;	//shift left **A**
0b11001	**A** = (**B** << **Literal**); **C** = **B**[31]; **V** = (**Literal** >= 32) ? 0:1;	//shift left **B**
0b11010	**A** = (**B** << 1); **C** = **B**[31]; **V** = 0;	//shift left **B**
0b11011	**n** = (**A** - **B**); **C** = (**A** - **B**)[32]; **V** = 0;	//compare
0b11100	**A** = (**A** >> **B**); **C** = **A**[0]; **V** = (**B** >= 32) ? 1:0;	//shift right **A**
0b11101	**A** = (**B** >> **Literal**); **C** = **A**[0]; **V** = (**Literal** >= 32) ? 1:0;	//shift right **B**
0b11110	**A** = (**B** >> 1); **C** = **A**[0]; **V** = 0;	//shift right **B**
0b11111	**n** = (**B** - **A**); **C** = (**B** - **A**)[32]; **V** = 0;	//compare

FIG. 46

OpdSel	SELECTED OPERANDs	
0b0000aaaaa	**File**	**File**@(**OpdSel**[4:0] | **FileBase**); Allows paged access to any part of the register file.
0b0001aaaaa	**CpuReg**	**File**@{2'b11, **CpuId**, **OpdSel**[4:0]}; Allows direct access to Cpu specific registers.
0b001XXXXXX	**reserved**	Reserved for future expansion.
0b0100000XX	**CpuStatus**	0b0000000000000BHD00000000000000CC This is a read-only register providing information about the Cpu executing (**OpdSel**[1:0]) cycles after the current cycle. "**CC**" represents a value indicating the Cpu. Currently, only **CpuId** values of 0, 1 and 2 are returned. "**H**" represents the current state of **Hlt**, "**D**" indicates **DbgMd** and "**B**" indicates **BigMd**. Writing this register has no effect.
0b0100001XX	**reserved**	Reserved for future expansion.
0b0100010XX	**Pc**	0x0000AAAA Writing to this address causes the program control logic to use **AluOut** as the new **Pc** value in the event of a **Jmp**, **Jcc** or **Jsr** instruction for the Cpu executing during the current cycle. If the current instruction is **Nxt**, **Map**, or **Rts**, the register write has no effect. Reading this register returns the value in **Pc** for the Cpu executing (**OpdSel**[1:0]) cycles after the current cycle.
0b0100011XX	**DbgAddr**	0xD000AAAA Writing to this register alters the contents of the debug address register (**DbgAddr**) for the Cpu executing (**OpdSel**[1:0]) cycles after the current cycle. **DbgAddr** provides the fetch address for the control-store when **DbgMd** has been selected and the Cpu is executing. **DbgAddr** is also used as the control-store address when performing a **WrWcs@DbgAddr** or **RdWcs@DbgAddr** operation. "**D**" represents bit 31 of the register. It is a general purpose flag that is used for event indication during simulation. Reading this register returns a value of 0x00000000.
0b01001XXXX	**reserved**	Reserved for future expansion.
0b010100000	**RamAddr** {0b1CCC, 0x000, 0b1, AAAA} **RamAddr** = **AluOut**[15] ? **AluOut** : (**AluOut** | **RamBase**); **PrevCC** = **AluOut**[31] ? CCC : **AluCC**; A read/write register. When reading this register, the Alu condition codes from the previous instruction are returned together with **RamAddr**.	

bit	name	description
31		Always 1.
30	**PrevC**	Previous Alu Carry.
29	**PrevV**	Previous Alu Overflow.
28	**PrevZ**	Previous Alu Zero.
27:16		Always 0.
15		Always 1.
14:0	**RamAddr**	Contents of last Sram address used.

When writing this register, if alu_out[31] is set, the previous condition codes will be overwritten with bits 30:28 of **AluOut**. If **AluOut**[15] is set, bits 14:0 will be written to the **RamAddr**. If **AluOut** [15] is not set, bits 14:0 will be ored with the contents of the **RamBase** and written to the **RamAddr**

FIG. 47

OpdSel	SELECTED OPERANDs
0b010100001	AddrRegA 0x0000AAAA

AddrRegA = AluOut;

A read/write operand which loads **AddrRegA** used to provide the address for read and write operations. When **AddrRegA**[15] is set, the contents will be presented directly to the ram. When **AddrRegA**[15] is reset, the contents will first be ored with the contents of the **RamBase** register before presentation to the ram. Writing to this register takes priority over Literal loads using **FlgOp**. Reading this register returns the current value of the register.

0b010100010 AddrRegB 0x0000AAAA

AddrRegB = AluOut;

A read/write operand which loads **AddrRegB** used to provide the address for read and write operations.
When **AddrRegB**[15] is set, the contents will be presented directly to the ram. When **AddrRegB**[15] is reset, the contents will first be ored with the contents of the **RamBase** register before presentation to the ram. Writing to this register takes priority over Literal loads using **FlgOp**. Reading this register returns the current value of the register.

0b010100011 AddrRegAb 0x0000AAAA
AddrRegA = AluOut; AddrRegB = AluOut;

A destination only operand which loads **AddrRegB** and **AddrRegA** used to provide the address for read and write operations Writing to this register takes priority over Literal loads using **FlgOp**. Reading this register returns the value 0x00000000.

0b010100100 **RamBase** 0x0000AAAA
RamBase = AluOut;

A read/write register which provides the base address for ram read and write cycles. When **RamAddr**[15] is set, the contents will not be used. When **RamAddr**[15] is reset, the contents will first be ored with the contents of the **RamBase** register before presentation to the ram. Reading this register returns the value for the current Cpu.

0b010100101 **FileBase** 0b00000000000000000000000AAAAAAAAA
FileBase = AluOut;
FileAddr = OpdSel[8] ? OpdSel:(OpdSel + FileBase);

A read/write register which provides the base address for **file** read and write cycles. When **OpdSel[8]** is set, the contents will not be used and **OpdSel** will be presented directly to the address lines of the file. When **OpdSel**[8] is reset, the contents will first be ored with the contents of the **FileBase** register before presentation to the file. Reading this register returns the value for the current Cpu.

0b010100110 **InstrRegL** 0xIIIIIIII

This is a read-only register which returns the contents of **InstrReg**[31:0]. Writing to this register has no effect.

0b010100111 **InstrRegH** 0x00IIIIII

This is a read-only register which returns the contents of **InstrReg**[55:32]. Writing to this register has no effect.

FIG. 48

OpdSel	SELECTED OPERANDs	
0b010101000	**Minus1**	0xffffffff This is a read-only register which supplies a value 0xffffffff.. Writing to this register has no effect.
0b010101001	**FreeTime**	A free-running timer with a resolution of 1.00 microseconds and a maximum count of 71 minutes. This timer is cleared during reset.
0b010101010	**LiteralL**	**Instr**[15:0] A read-only register. Writing to this register has no effect
0b010101011	**LiteralH**	**Instr**[15:0]<<16; A read-only register. Writing to this register has no effect

0b010101100 **MacData** - Writing to this address loads the **AluOut** data into the **MacData** register for use during Mac operations. The Mac operation, resulting from writing to the **MacOp** register, determines the definition of the **MacData** register contents as follows.

MacOp	MacData definition
Mstop	0bXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX **MacData** is not used for the **StopM** operation.
WrMcfg	**hrstl**, rsvd, rsvd, **crcen, fulld, hrstl, hugen, nopre, paden, prtyl, xdl10, ipgr1[6:0]**, **ipgr2[6:0]**, **ipgt[6:0]**. Loads the **MacCfg** register with the contents of the **MacData** register. Refer to LSI Logic's *Ethernet-110 Core Technical Manual* for detailed definitions of these bits.
WrMrng	0bXXXXXXXXXXXXXXXXXXXXXXSSSSSSSSSS Loads **seed**[10:0] into the Mac's random number generator.
RdPhy	0bXXXXRRRRXXXXPPPPXXXXXXXXXXXXXXXX Reads register[R] of phy[P].
WrPhy	0bXXXXRRRRXXXXPPPPDDDDDDDDDDDDDDDD Writes register[R] of phy[P] with **MacData**[15:0].

Reading this register returns **prsd[15:0]** of **Mac0** which contains phy status data returned to the Mac at the completion of a **RdPhy** command. This data is invalid while **MacBsy** is asserted as a result of a **RdPhy** command. Refer to the appropriate phy technical manual for a definition of the phy register contents.

FIG. 49

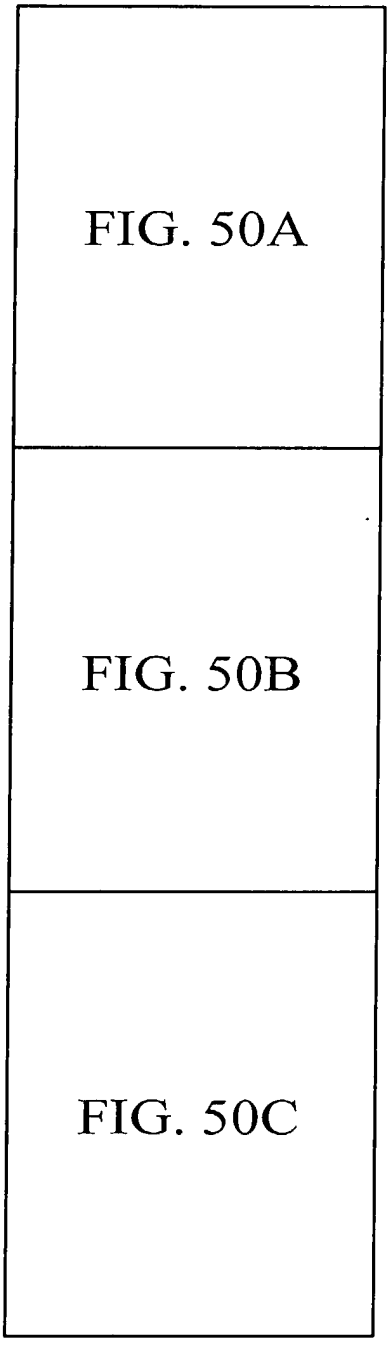


FIG. 50

OpdSel	SELECTED OPERANDs

0b010101101 **MacOp** - A write only register. Writing to this address loads the **MacSel** register and staRts execution of the specified operation as follows.

AluOut	description
0xXXXXX0XM	**Mstop** - Halts execution of a **MacOp** for **Mac**[M]. The user must wait for **MacBsy** to be deasserted before issuing another command or changing the contents of **MacData**.
0xXXXXX1XM	**WrMcfg** - Writes the contents of **MacData** to the **MacCfg** register of **Mac**[M]. The user must wait for **MacBsy** to be deasserted before issuing another command or changing the contents of **MacData**.
0xXXXXX2XM	**WrMrng** - Writes the contents of **MacData** to the **seed** register of **Mac**[M]. The user must wait for **MacBsy** to be deasserted before issuing another command or changing the contents of **MacData**.
0xXXXXX3XM	**RdPhy** - Reads the contents of reg[R] for phy[P] on the MII management bus of Mac[M]. The contents may be read from **MacData** after **MacBsy** has been de-asserted.
0xXXXXX4XM	**WrPhy** - Writes the contents of **MacData**[15:0] to e reg[R] of phy[P] on the MII management bus of **Mac**[M]. The user must wait for **MacBsy** to be deasserted before issuing another command or changing the contents of **MacData**.
0xXXXXX8XM	**WrAddrAL** - Writes the contents of **MacData**[15:0] to **MacAddrA[15:0]** for **Mac**[M].
0xXXXXX9XM	**WrAddrAH** - Writes the contents of **MacData**[11:0] to **MacAddrA[47:16]** for **Mac**[M].
0xXXXXXaXM	**WrAddrBL** - Writes the contents of **MacData**[15:0] to **MacAddrB[15:0]** for **Mac**[M].
0xXXXXXbXM	**WrAddrBH** - Writes the contents of **MacData**[11:0] to **MacAddrB[47:16]** for **Mac**[M].

b010101110 **ChCmd** A write-only register.

bit	name	description
31:11	reserved	Data written to these bits is ignored.
10:8	**command**	0 - Stops execution of the current operation and clears the corresponding event flag. 1 - Transfer data from ExtMem to ExtMem. 2 - Transfer data from Pci to ExtMem. 3 - Transfer data from ExtMem to Pci. 4 - Transfer data from Sram to ExtMem. 5 - Transfer data from ExtMem to Sram. 6 - Transfer data from Pci to Sram. 7 - Transfer data from Sram to Pci.
07:05	reserved	Data written to these bits is ignored.
04:00	**ChId**	Provides the channel number for the channel command.

FIG. 50A

0b010101110 **ChEvnt** A read-only register.

bit	name	description
31:00	**ChDn**	Each bit represents the done flag for the respective dma channel. These bits are set by a dma sequencer upon completion of the channel command. Cleared when the processor writes 0 to the corresponding **ChCmd** register.

0b010101111 **GenEvnt** A read-only register.

bit	name	description
31	**PciRdEvnt**	Indicates that a PCI initiator is attempting to read a mproc. register.
30	**PciWrEvnt**	Indicates that a PCI initiator has posted a write to a mproc. register.
29	**TimeEvnt**	An event which occurs once every 2.00 milliseconds.
28:00	**reserved**	Reserved for future use.

0b010110000 **QCtrl** A write-only register used to select and manipulate a **Q**.

bit	name	description
31:11	reserved	Data written to these bits are ignored.
10:8	**QSz**	Used only during **InitQ** operations to specify the size of the **QBdy** in Dram. 7 – Queue depth is 32K entries (128KB). 6 – Queue depth is 16K entries (64KB). 5 – Queue depth is 8K entries (32KB). 4 – Queue depth is 4K entries (16KB). 3 – Queue depth is 2K entries (8KB). 2 – Queue depth is 1K entries (4KB). 1 – Queue depth is 512 entries (2KB). 0 – Queue depth is 256 entries (1KB).
7:5	**QOp**	Specifies the queue operation to perform. 7 – **DblQ** Disables all queues. 6 – **EnQ** Enables all queues. 5 – **RdBdy** Increments the **QBdyRdPtr** and increments the **QTlWrPtr**. 4 – **WrBdy** Decrements the **QBdyWrPtr** and increments the **QHdRdPtr**. 3 – **RdQ** Returns a queue entry in register **QData**.. 2 – **rsvd** Reserved. Not to be used. 1 – **InitQ** Set the queue status to empty and initializes **QSz**. 0 – **SelQ** Selects the **QId** to be utilized during writes to **QData**.

FIG. 50B

4:0	**QId**	Specifies the queue on which to perform all operations except **DblQ** or **EnQ**.
0b010110001	**QData**	A read/write register. Writing this register will result in the data being pushed on to the selected queue. Reading this register fetches queue data popped off during the previous **RdQ** operation.
0b010110010	**reserved**	Reserved for future expansion.
0b010110011	**XcvCtrl**	A write-only register used to enable and disable Mac transmit and receive sub-channels.

bit	name	description
31:09	reserved	Data written to these bits are ignored.
8	**enable**	When set, indicates to the Mac transmit or receive sequencer that the subchannel contains a transmit or receive descriptor.
07:05	reserved	Data written to these bits is ignored.
04	**RcvCh**	Selects a Mac receive subchannel when set. Selects a Mac transmit subchannel when cleared.
03	reserved	Data written to this bit are ignored.
02	**SubCh**	Selects subchannel B when set or A when reset.
01:00	**MacId**	Provides the Mac number for the subchannel enable bit.

0b010110100	**Lru**	0x0000000A A read/write operand indicating which of the 16 entries is least recently used. When Reading This register the least recently used entry is returned, after which it is automatically made the most recently used entry. This register should only be read in conjunction with a 'Move' operation of the ALU, else the results are unpredictable. Writing to this register forces the addressed entry to become the least recently used entry.
0b010110101	**Mru**	0x0000000A A write only operand forcing the addressed entry to become the most recently used entry.
0b010111000	**QInRdy**	A read-only register comprising **QHd** not full flags for each of the 32 queues.
0b010111001	**QOutRdy**	A read-only register comprising **QTl** not empty flags for each of the 32 queues.
0b010111010	**QEmpty**	A read-only register comprising **QEmpty** flags for each of the 32 queues.
0b010111011	**QFull**	A read-only register comprising **QFull** flags for each of the 32 queues.
0b0101111XX	**reserved**	Reserved for future expansion.
0b0110XXXXX	**Constants**	{0b000, **OpdSel**[4:0]}
0b01110XXXX	**reserved**	Reserved for future expansion.

FIG. 50C

OpdSel	SELECTED OPERANDs
0b01111XXXX	Sram OPERATIONS

OpdSel[3]	PostAddrOp
0	nop
1	RamAddr = RamAddr + (OpdSel[1:0]);

OpdSel[2]	transpose_Ctrl
0	don't transpose
1	transpose bytes

OpdSel[1:0]	RamOpdSz
0	quadlet
1	triplet
2	doublet
3	byte

RAM READ ATTRIBUTES				SOURCE OPERAND			
endian mode	trans-pose	byte offs	Sram data	sz=Q	sz=T	sz=D	sz=B
little	0	0	abcd	abcd	0bcd	00cd	000d
little	0	1	abcX	trap	0abc	00bc	000c
little	0	2	abXX	trap	trap	00ab	000b
little	0	3	aXXX	trap	trap	trap	000a
little	1	0	abcd	dcba	0dcb	00dc	000d
little	1	1	abcX	trap	0cba	00cb	000c
little	1	2	abXX	trap	trap	00ba	000b
little	1	3	aXXX	trap	trap	trap	000a
BIG	0	0	abcd	abcd	0abc	00ab	000a
BIG	0	1	Xbcd	trap	0bcd	00bc	000b
BIG	0	2	XXcd	trap	trap	00cd	000c
BIG	0	3	XXXd	trap	trap	trap	000d
BIG	1	0	abcd	dcba	0cba	00ba	000a
BIG	1	1	Xbcd	trap	0dcb	00cb	000b
BIG	1	2	XXcd	trap	trap	00dc	000c
BIG	1	3	XXXd	trap	trap	trap	000d

RAM WRITE ATTRIBUTES				SOURCE OPERAND			
endian mode	trans-pose	Opd size	Alu out	OF=0	0F=1	OF=2	OF=3
little	0	Q	abcd	abcd	trap	trap	trap
little	0	T	Xbcd	-bcd	bcd-	trap	trap
little	0	D	XXcd	--cd	-cd-	cd--	trap
little	0	B	XXXd	---d	--d-	-d--	d---
little	1	Q	abcd	dcba	trap	trap	trap
little	1	T	Xbcd	-dcb	dcb-	trap	trap
little	1	D	XXcd	--dc	-dc-	dc--	trap
little	1	B	XXXd	---d	--d-	-d--	d---
big	0	Q	abcd	abcd	trap	trap	trap
big	0	T	Xbcd	bcd-	-bcd	trap	trap
big	0	D	XXcd	cd--	-cd-	--cd	trap
big	0	B	XXXd	d---	-d--	--d-	---d
big	1	Q	abcd	dcba	trap	trap	trap
big	1	T	Xbcd	dcb-	-dcb	trap	trap
big	1	D	XXcd	dc--	-dc-	--dc	trap
big	1	B	XXXd	d---	-d--	--d-	---d

0b1aaaaaaaa	File	File@OpdSel[8:0]; Allows direct, non-paged, access to the top half of the register file.

FIG. 51

TstSel	SELECTED TEST	
0bX00XXXXX	**Tst** = **TstSel**[7] ^ **AluOut**[**TstSel**[4:0]]	//Alu bit
0bX0100000	**Tst** = **TstSel**[7] ^ **C**	//carry
0bX0100001	**Tst** = **TstSel**[7] ^ **V**	//error
0bX0100010	**Tst** = **TstSel**[7] ^ **Z**	//zero
0bX0100011	**Tst** = **TstSel**[7] ^ (**Z** | ~**C**)	//less or equal
0bX0100100	**Tst** = **TstSel**[7] ^ **PrevC**	//previous carry
0bX0100101	**Tst** = **TstSel**[7] ^ **PrevV**	//previous error
0bX0100110	**Tst** = **TstSel**[7] ^ **PrevZ**	//previous zero
0bX0100111	**Tst** = **TstSel**[7] ^ (**PrevZ** & **Z**)	//64b zero
0bX0101000	**Tst** = **TstSel**[7] ^ **QOpDn**	//queue op okay
0bX0101001	**Tst** = reserved	
0bX010101X	**Tst** = reserved	
0bX01011XX	**Tst** = reserved	
0bX0110XXX	**Tst** = **TstSel**[7] ^ **Lock**[**TstSel**[2:0]] **Lock**(**TstSel**[2:0]) = 1;	//tests the current value of //the Lock then set it.
0bX0111XXX	**Tst** = **TstSel**[7] ^ **Lock**[**TstSel**[2:0]]	//tests the value of Lock.
0bX01XXXXX	**Tst** = reserved	
0bX1XXXXXX	**Tst** = reserved	

FIG. 52

FlgSel	FLAG OPERATION	
0b00000000	No operation.	
0b00000001	**SelfRst**	Forces a self reset for the entire chip excluding the PCI configuration registers
0b00000010	**SelBigEnd**	Selects big-endian mode for ram accesses for the current Cpu.
0b00000011	**SelLitEnd**	Selects little-endian mode for ram accesses for the current Cpu.
0b00000100	**DblMap**	Disable instruction re-mapping for the current Cpu.
0b00000101	**EnbMap**	Enable instruction re-mapping for the current Cpu.
0b0000011X	reserved	
0b00001XXX	reserved	
0b00010XXX	**ClrLck**	**Lock[FlgSel[2:0]] = 0;** Clears the semaphore register bit for the current Cpu only.
0b00011XXX	reserved	
0b0010XXXX	**AddrOp**	

FlgSel[3.2]	AddrSelect	
0	**RamAddr = Literal[15]**	**? Literal : (Literal | RamBase);**
1	**RamAddr = AddrRegA[15]**	**? AddrRegA : (AddrRegA | RamBase),**
2	**RamAddr = AddrRegB[15]**	**? AddrRegB : (AddrRegB | RamBase);**
3	**if (OpdA == RamAddr)**	
	RamAddr = AluOut[15]	**? AluOut : (AluOut | RamBase);**
	else if (OpdA == ram)	
	RamAddr = AddrRegB[15]	**? AddrRegB : (AddrRegB | RamBase);**
	else	
	RamAddr = AddrRegA[15]	**? AddrRegA : (AddrRegA | RamBase);**

FlgSel[1.0]	addr reg load	
0	**nop**	
1	**AddrRegA = Literal,**	
2	**AddrRegB = Literal;**	
3	**AddrRegA = Literal;**	**AddrRegB = Literal;**

note: When specifying the same register for both the load and select fields, the current value of the register, before it is loaded with the new value, will be used for the ram address.

0b0011XXXX	reserved	
0b01000000	**WrWcsL@Dbg**	Causes the bits [31:0] of the control-store at address **DbgAddr** to be written with the current **AluOut** data.
0b01000001	**WrWcsH@Dbg**	Causes the bits [63:32] of the control-store at address **DbgAddr** to be written with the current **AluOut** data then increments **DbgAddr**.
0b01000010	**RdWcsL@Dbg**	Causes the bits [31:0] of the control-store at address **DbgAddr** to be moved to file address 0x1ff
0b01000011	**RdWcsH@Dbg**	Causes the bits [63:32] of the control-store at address **DbgAddr** to be moved to file address 0x1ff then increments **DbgAddr**.
0b01000100	reserved	
0b010001XX	**Step**	Allows the Cpu (**FlgSel**[1:0]) cycles after the current cycle to execute a single instruction. There is no effect if the Cpu is not halted. An offset of 0 is not allowed.
0b010010XX	**PcMd**	Selects the **Pc** as the address source for the control-store during instruction fetches for the Cpu (**FlgSel**[1:0]) cycles after the current cycle.
0b010011XX	**DbgMd**	Selects the **DbgAddr** address register as the address source for the control-store during instruction fetches for the Cpu (**FlgSel**[1:0]) cycles after the current cycle.
0b010100XX	**Hlt**	Halts the Cpu (**FlgSel**[1:0]) cycles after the current cycle.
0b010101XX	**Run**	Clears Halt for the Cpu (**FlgSel**[1:0]) cycles after the current cycle.
0b01011XXX	reserved	
0b011XXXXX	reserved	
0b1XXXXXXX	reserved	

FIG. 53

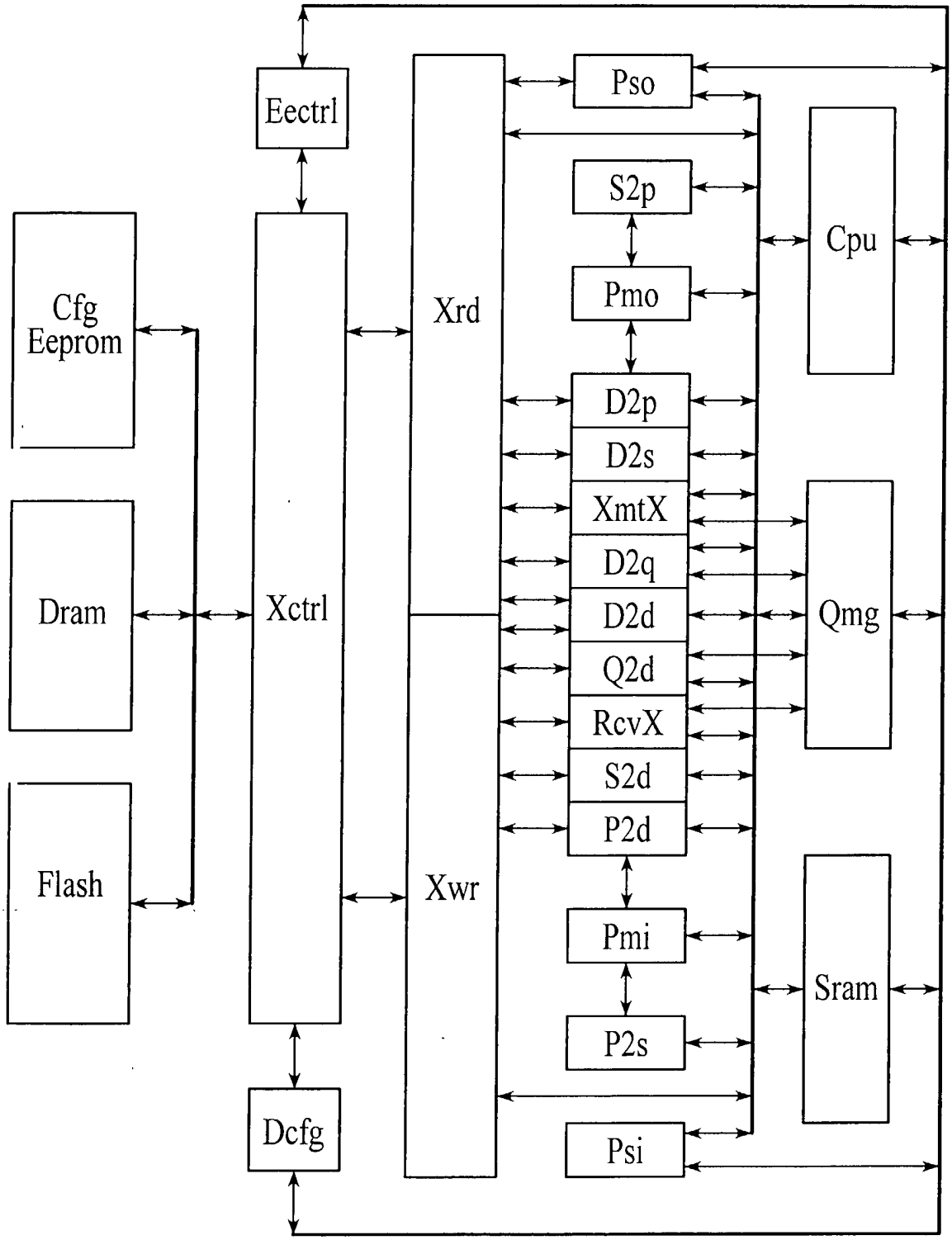

Eectrl
Cfg
Eeprom
Dram
Flash
Xctrl
Dcfg
Xrd
Xwr
Pso
S2p
Pmo
D2p
D2s
XmtX
D2q
D2d
Q2d
RcvX
S2d
P2d
Pmi
P2s
Psi
Cpu
Qmg
Sram

FIG. 54

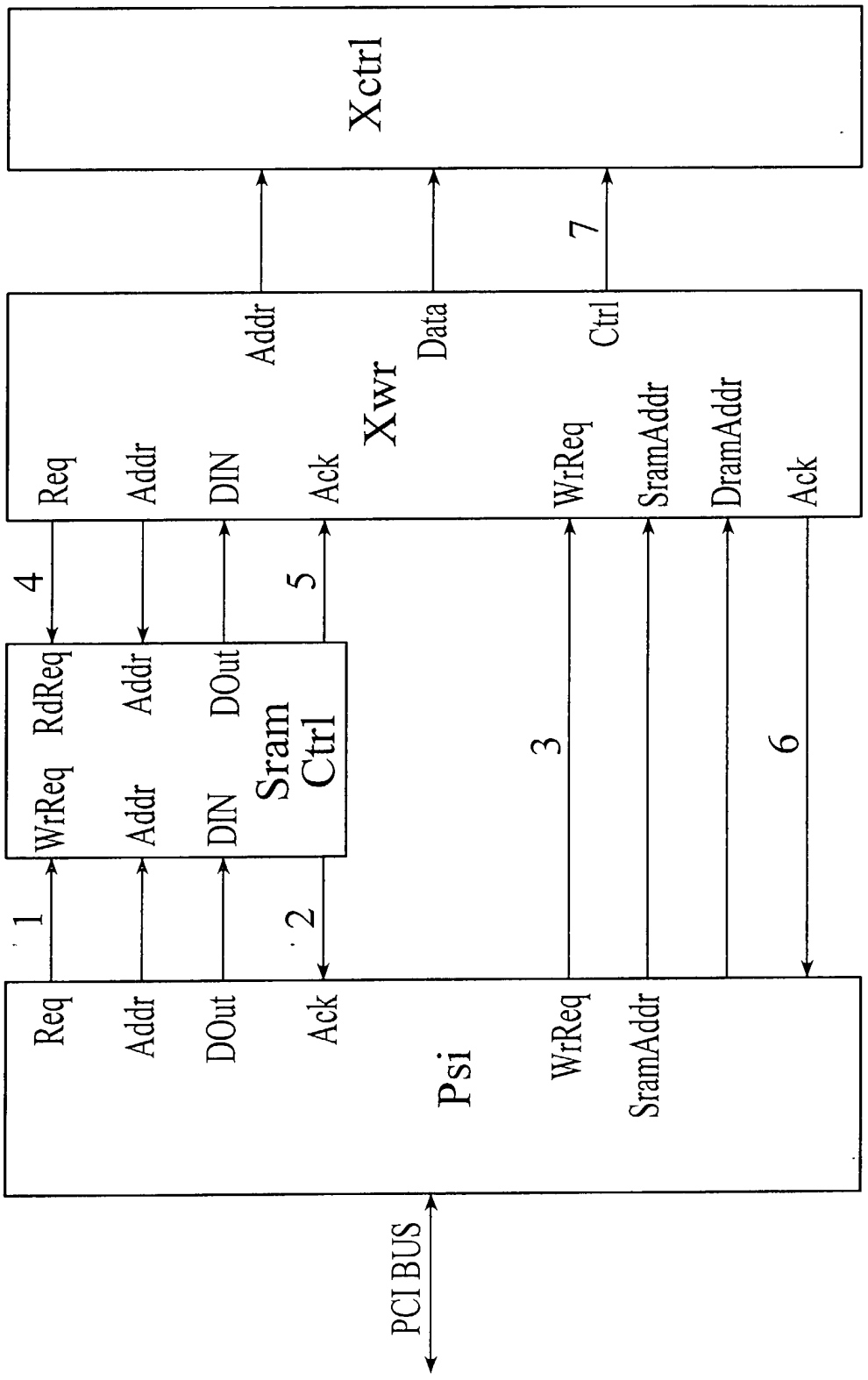

Xctrl
Xwr
Req
Addr
DIN
Ack
WrReq
SramAddr
DramAddr
Ack
Addr
Data
Ctrl
7
4
5
RdReq
Addr
DOut
WrReq
Addr
DIN
Sram
Ctrl
3
6
1
2
Req
Addr
DOut
Ack
Psi
WrReq
SramAddr
PCI BUS

FIG. 55

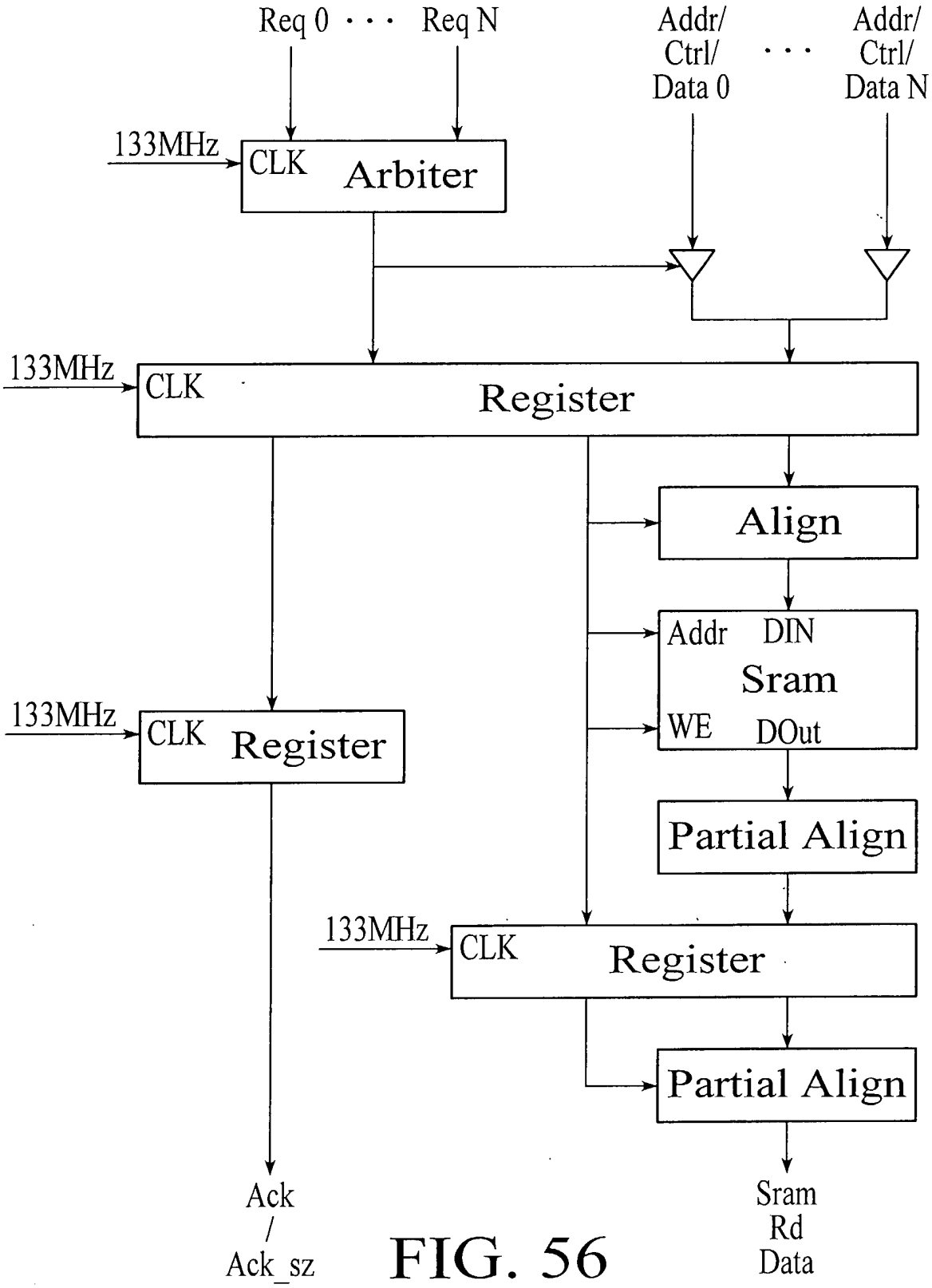

Req 0 · · · Req N
Addr/ Ctrl/ Data 0
· · ·
Addr/ Ctrl/ Data N
133MHz
CLK
Arbiter
133MHz
CLK
Register
Align
Addr
DIN
Sram
WE
DOut
133MHz
CLK
Register
Partial Align
133MHz
CLK
Register
Partial Align
Ack / Ack_sz
Sram Rd Data

FIG. 56

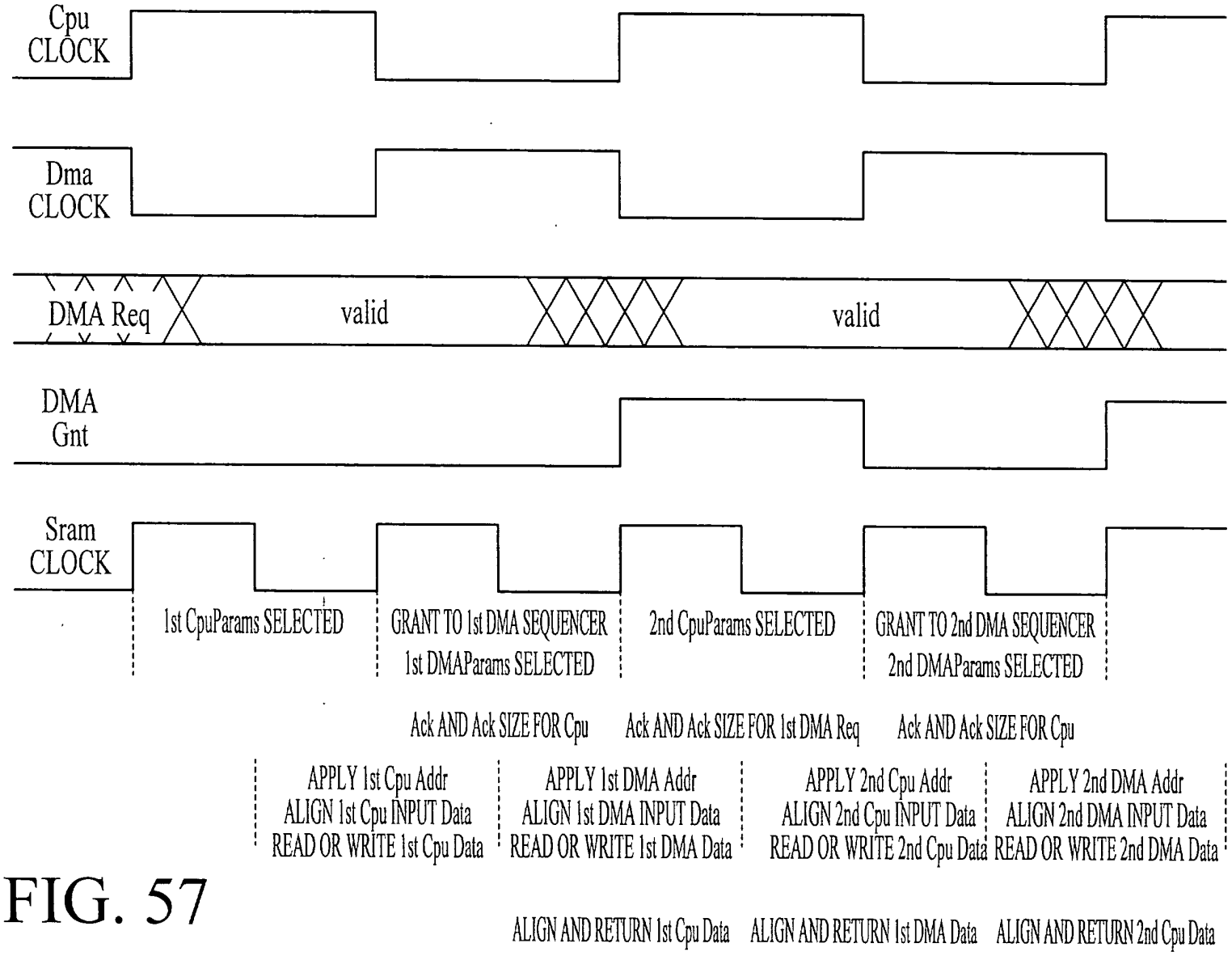


FIG. 57

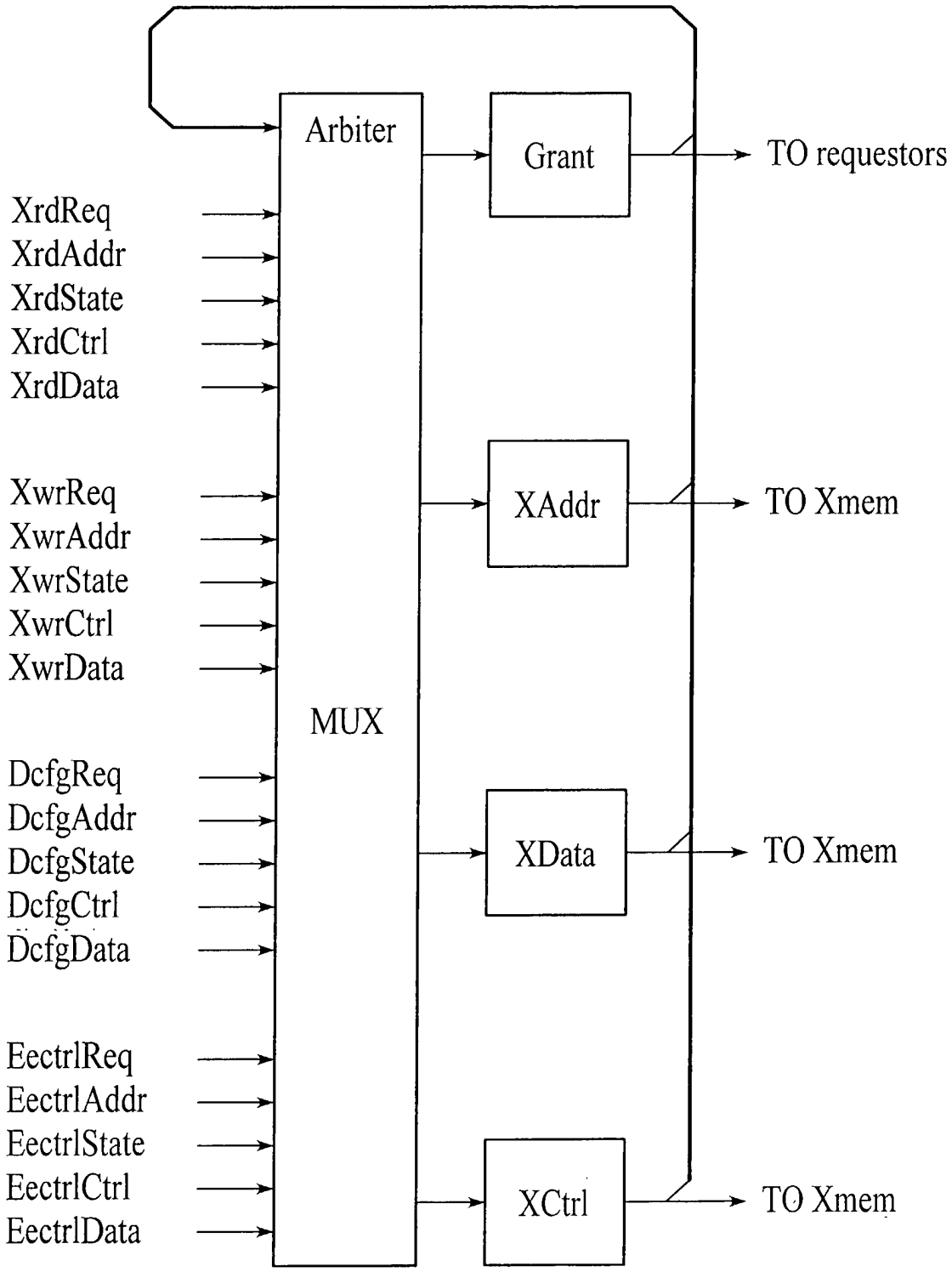

Arbiter
Grant
TO requestors
XrdReq
XrdAddr
XrdState
XrdCtrl
XrdData
XwrReq
XwrAddr
XwrState
XwrCtrl
XwrData
XAddr
TO Xmem
MUX
DcfgReq
DcfgAddr
DcfgState
DcfgCtrl
DcfgData
XData
TO Xmem
EectrlReq
EectrlAddr
EectrlState
EectrlCtrl
EectrlData
XCtrl
TO Xmem

FIG. 58

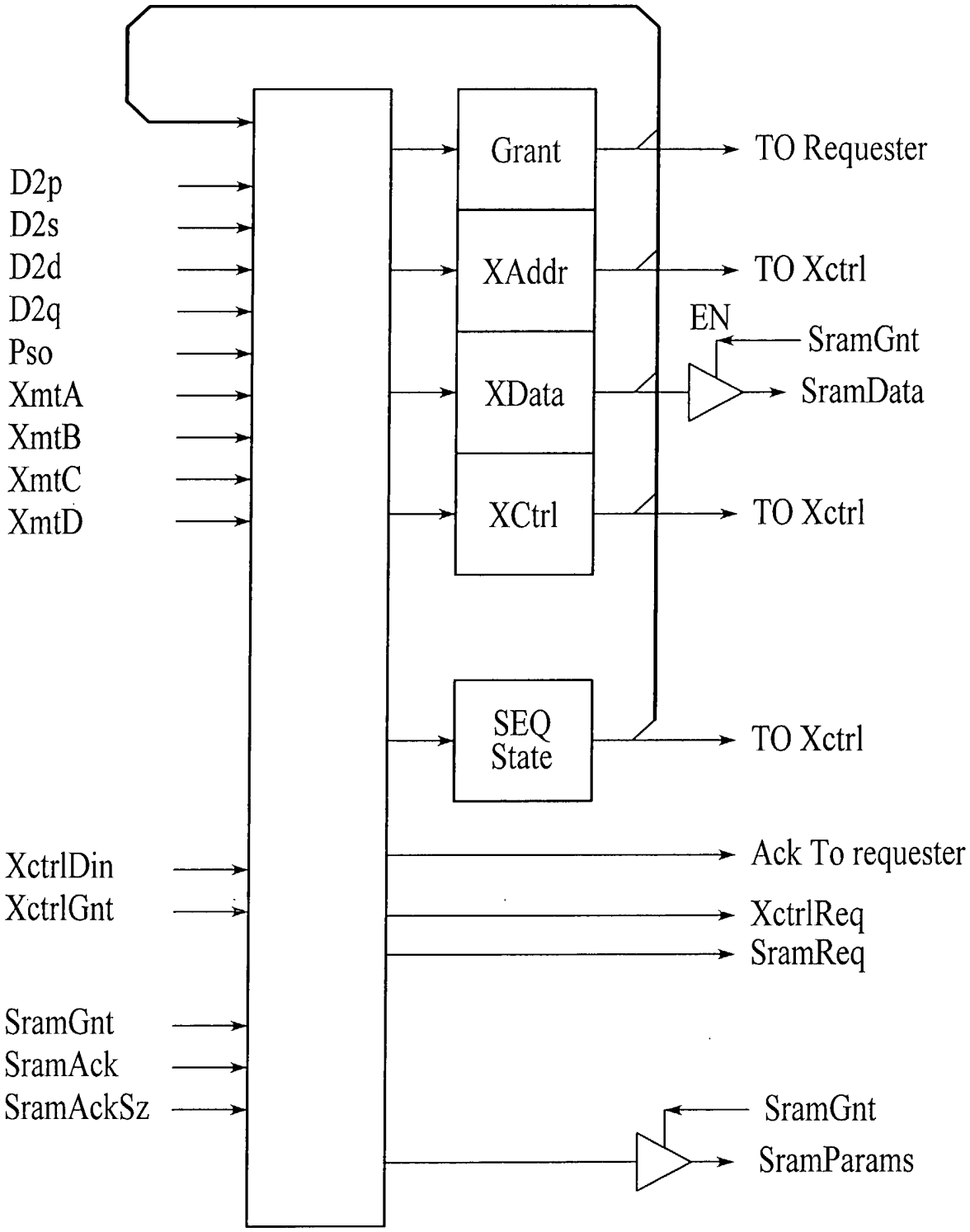

Grant
XAddr
XData
XCtrl
SEQ State
TO Requester
TO Xctrl
EN
SramGnt
SramData
TO Xctrl
TO Xctrl
D2p
D2s
D2d
D2q
Pso
XmtA
XmtB
XmtC
XmtD
XctrlDin
XctrlGnt
SramGnt
SramAck
SramAckSz
Ack To requester
XctrlReq
SramReq
SramGnt
SramParams

FIG. 59

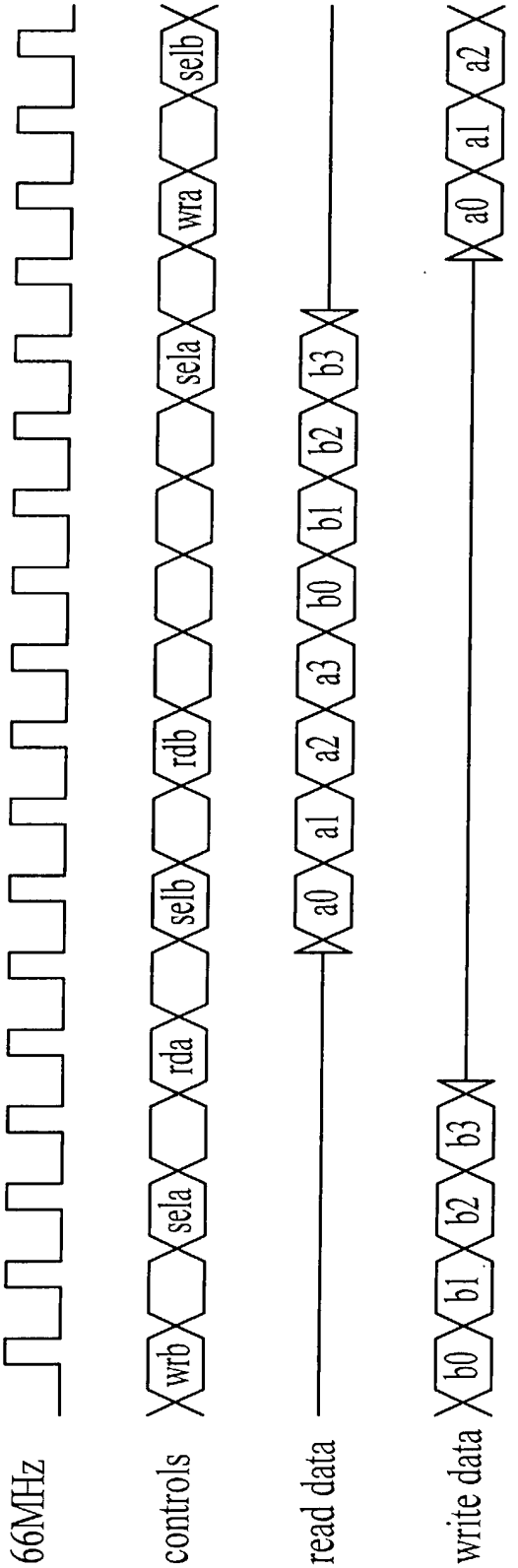

66MHz
controls
wrb
sela
rda
selb
rdb
sela
wra
selb
read data
a0
a1
a2
a3
b0
b1
b2
b3
write data
b0
b1
b2
b3
a0
a1
a2

FIG. 60

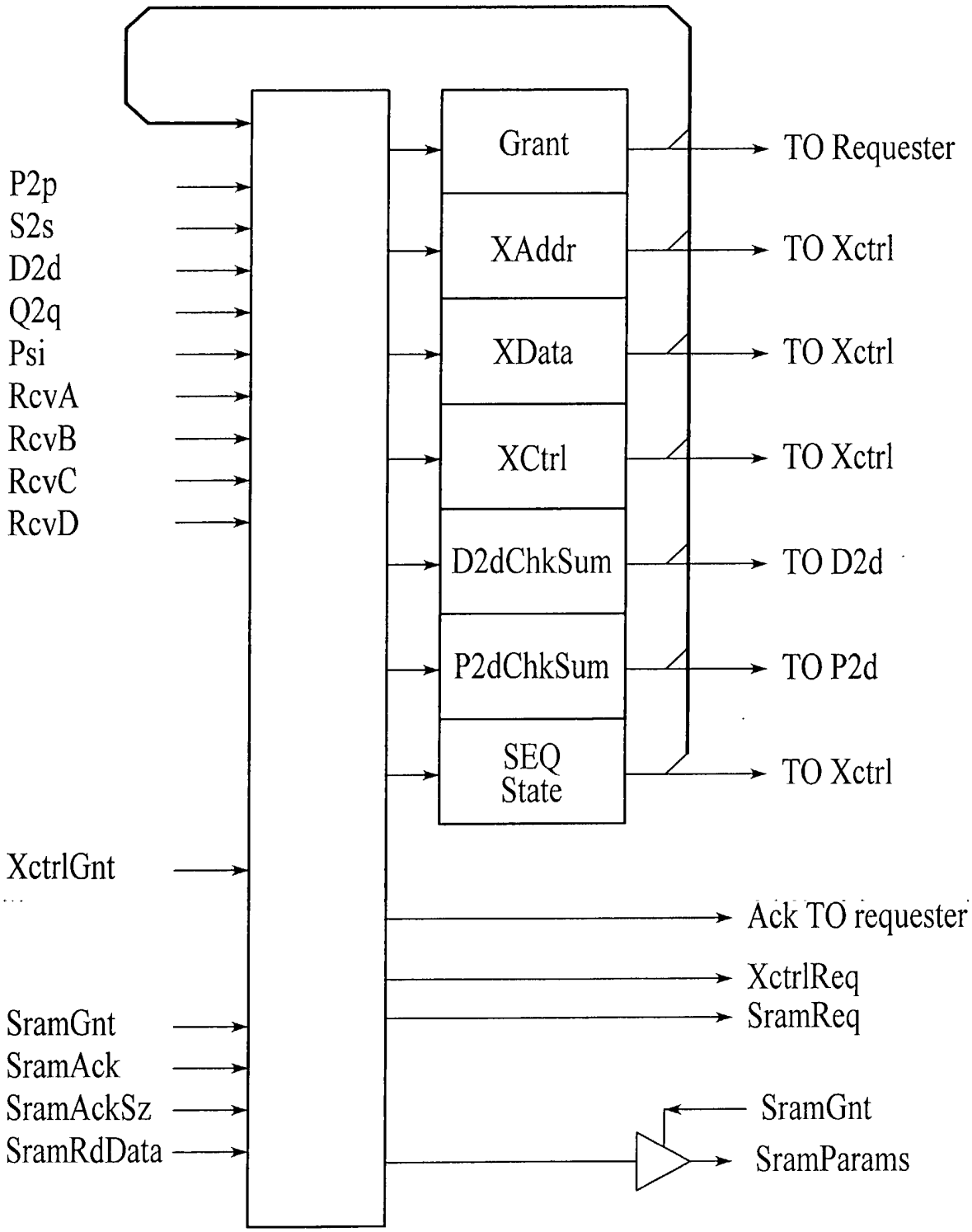

Grant
XAddr
XData
XCtrl
D2dChkSum
P2dChkSum
SEQ State
TO Requester
TO Xctrl
TO Xctrl
TO Xctrl
TO D2d
TO P2d
TO Xctrl
P2p
S2s
D2d
Q2q
Psi
RcvA
RcvB
RcvC
RcvD
XctrlGnt
SramGnt
SramAck
SramAckSz
SramRdData
Ack TO requester
XctrlReq
SramReq
SramGnt
SramParams

FIG. 61

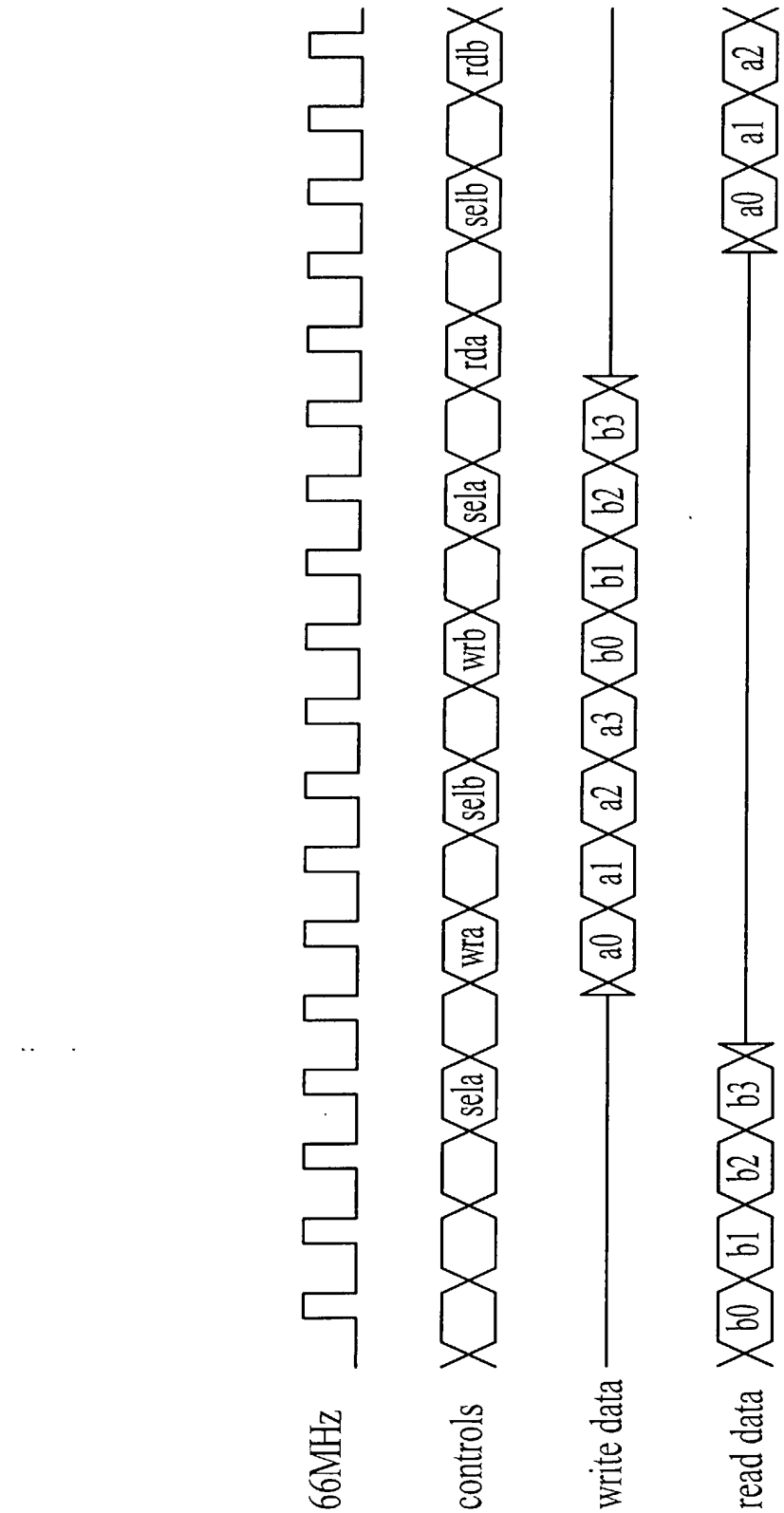

66MHz
controls
write data
read data
sela
wra
selb
wrb
sela
rda
selb
rdb
a0
a1
a2
a3
b0
b1
b2
b3
b0
b1
b2
b3
a0
a1
a2

FIG. 62

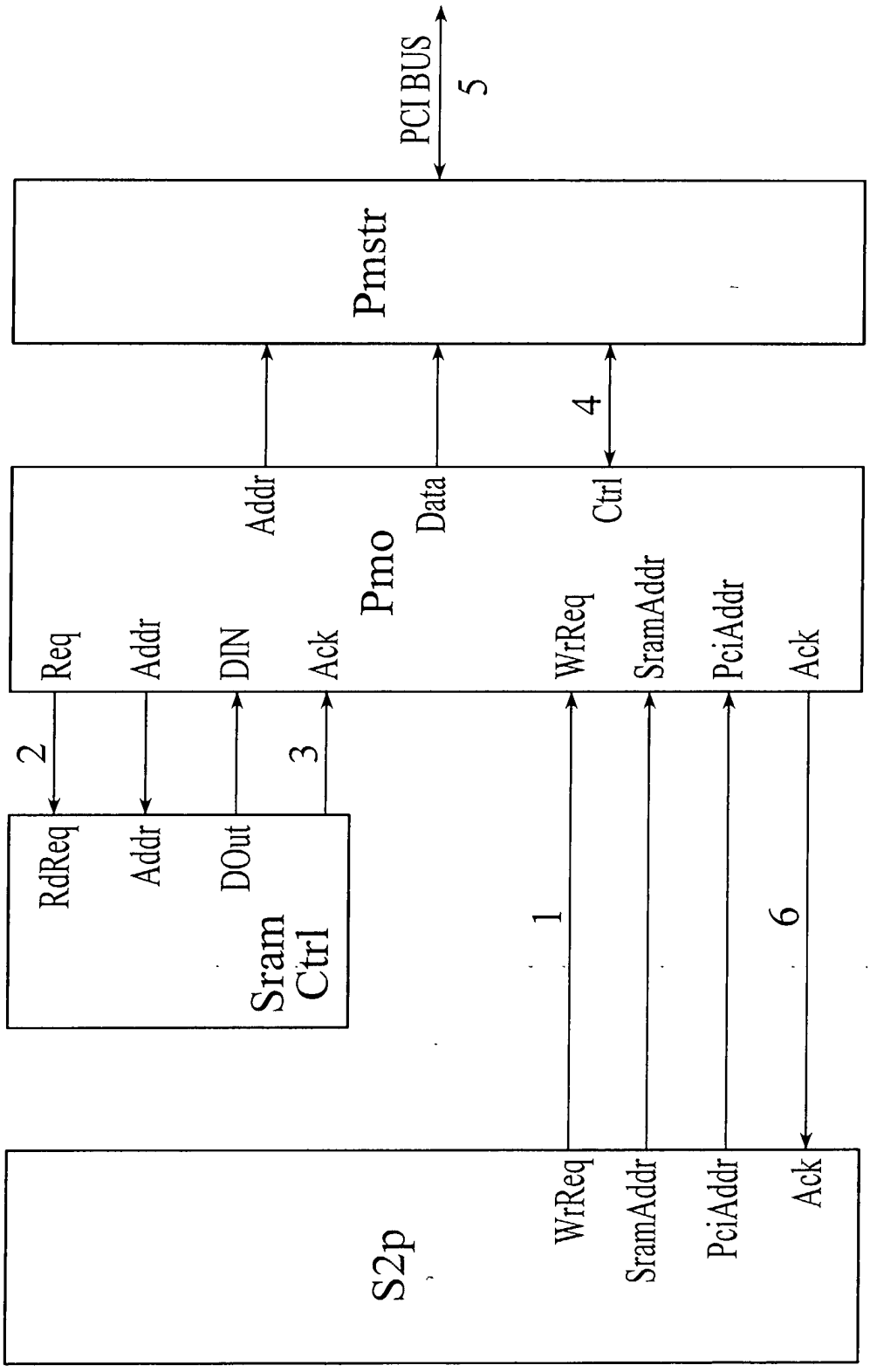

PCI BUS
5
Pmstr
4
Addr
Data
Ctrl
Pmo
Req
Addr
DIN
Ack
WrReq
SramAddr
PciAddr
Ack
2
3
RdReq
Addr
DOut
Sram Ctrl
1
6
S2p
WrReq
SramAddr
PciAddr
Ack

FIG. 63

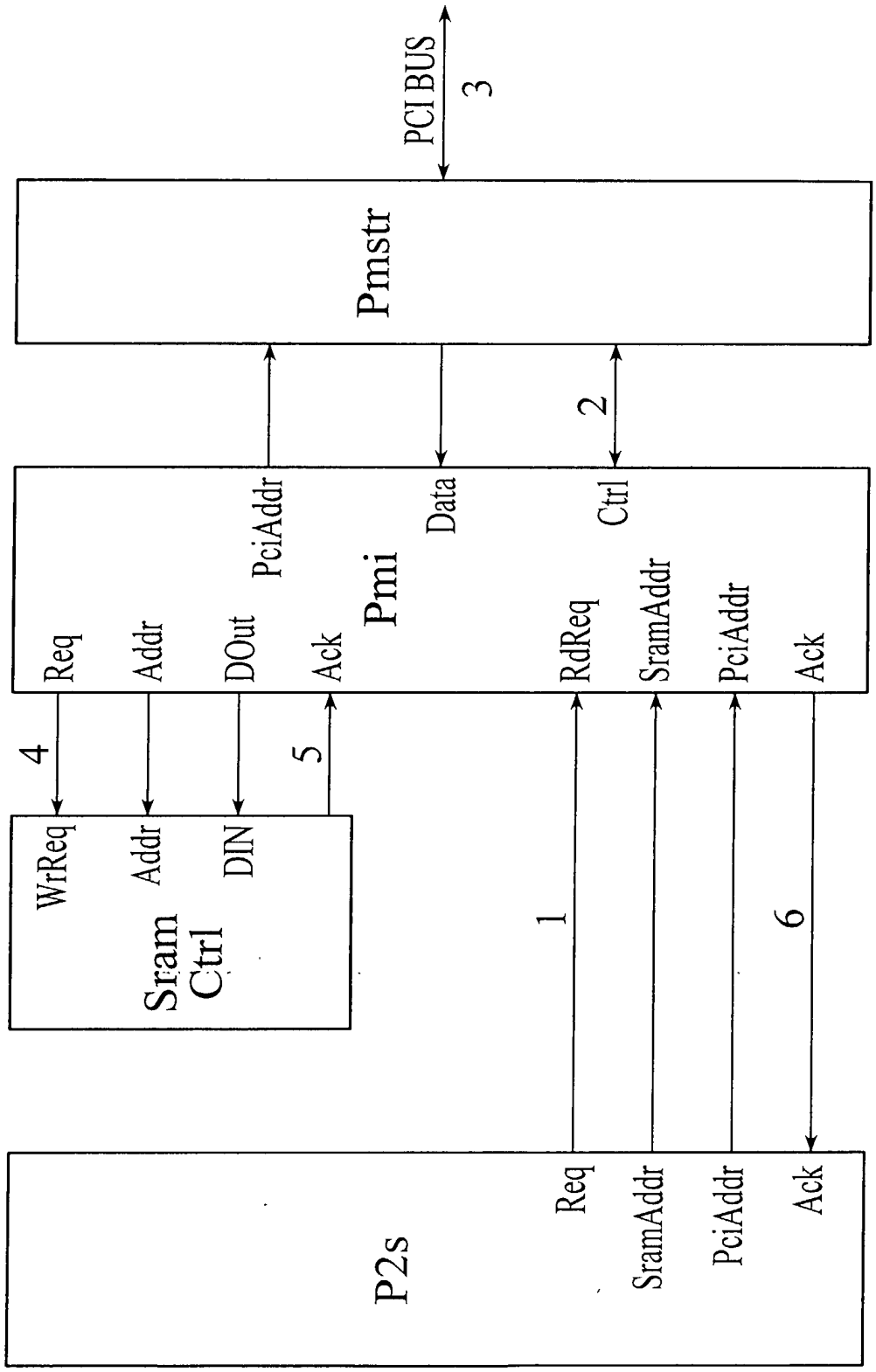

PCI BUS
3
Pmstr
2
PciAddr
Data
Ctrl
Req
Addr
DOut
Ack
Pmi
RdReq
SramAddr
PciAddr
Ack
4
5
WrReq
Addr
DIN
Sram
Ctrl
1
6
Req
SramAddr
PciAddr
Ack
P2s

FIG. 64

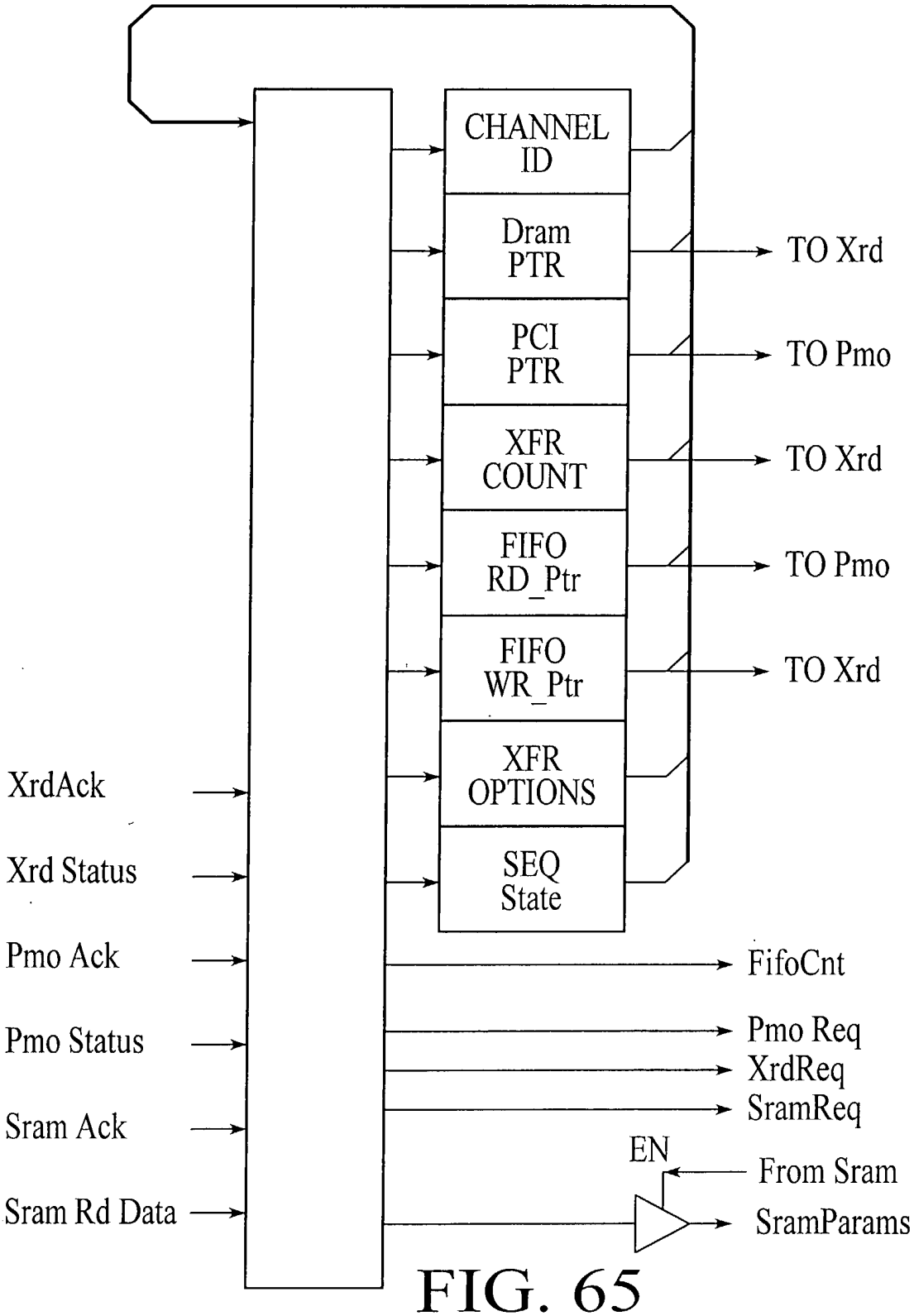

CHANNEL ID
Dram PTR
PCI PTR
XFR COUNT
FIFO RD_Ptr
FIFO WR_Ptr
XFR OPTIONS
SEQ State
TO Xrd
TO Pmo
TO Xrd
TO Pmo
TO Xrd
XrdAck
Xrd Status
Pmo Ack
Pmo Status
Sram Ack
Sram Rd Data
FifoCnt
Pmo Req
XrdReq
SramReq
EN
From Sram
SramParams

FIG. 65

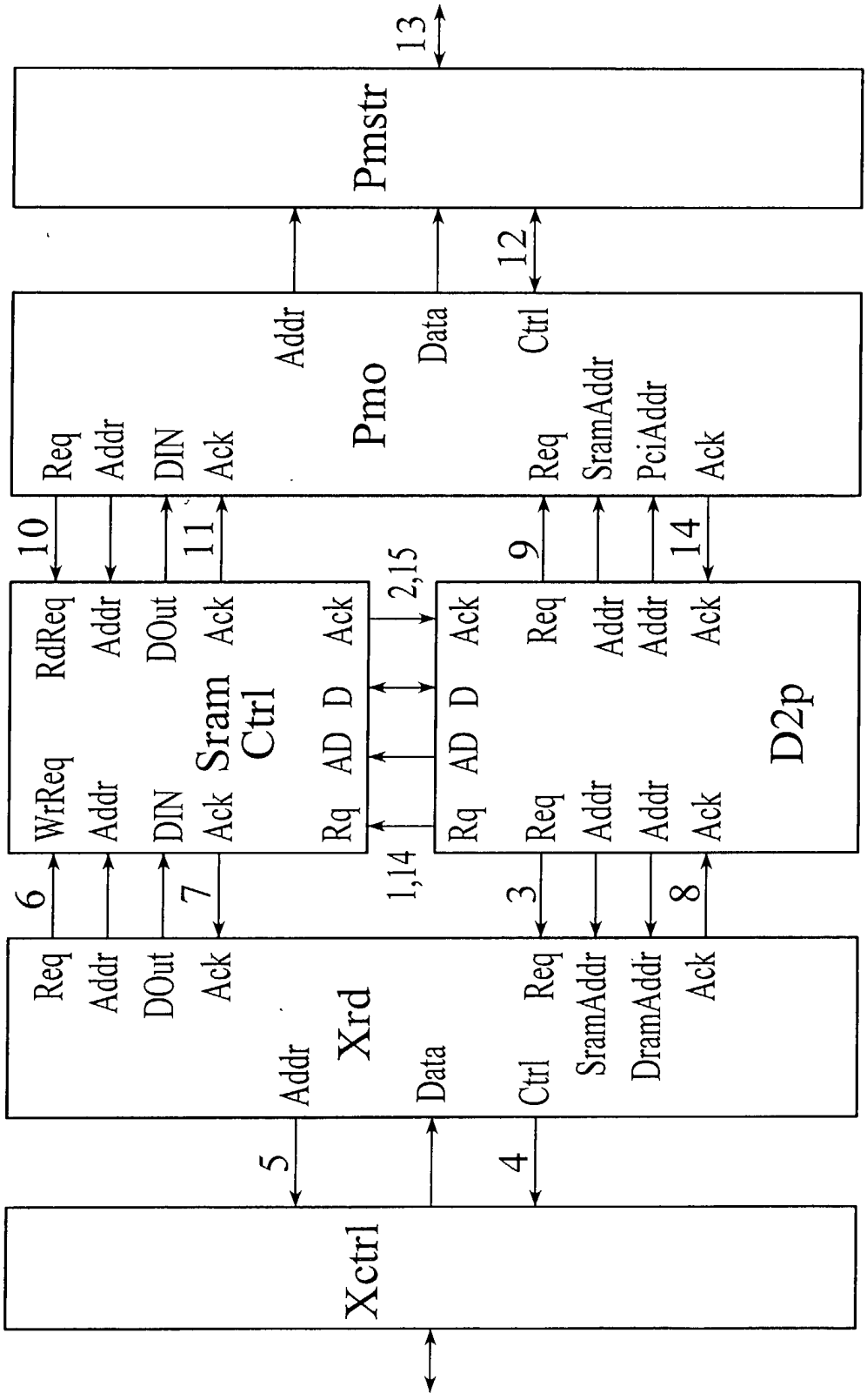

Pmstr
13
Pmo
Addr
Data
Ctrl
12
Req
Addr
DIN
Ack
Req
SramAddr
PciAddr
Ack
10
11
9
14
Sram Ctrl
RdReq
Addr
DOut
Ack
WrReq
Addr
DIN
Ack
Rq
AD
D
Ack
2,15
1,14
D2p
Req
Addr
Addr
Ack
Rq
AD
D
Ack
Req
Addr
Addr
Ack
6
7
3
8
Xrd
Req
Addr
DOut
Ack
Addr
Data
Ctrl
Req
SramAddr
DramAddr
Ack
5
4
Xctrl

FIG. 66

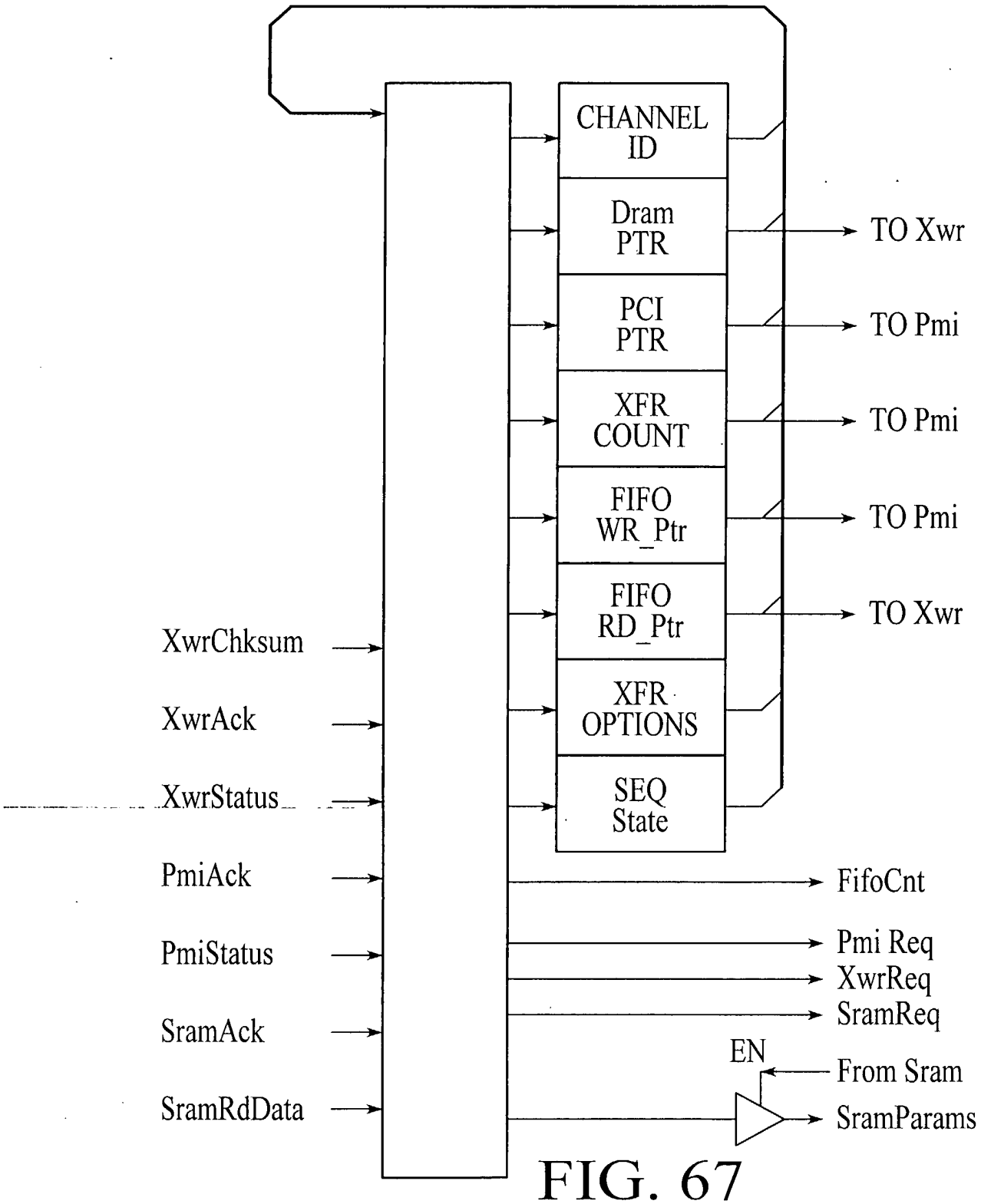

CHANNEL ID
Dram PTR
PCI PTR
XFR COUNT
FIFO WR_Ptr
FIFO RD_Ptr
XFR OPTIONS
SEQ State
TO Xwr
TO Pmi
TO Pmi
TO Pmi
TO Xwr
XwrChksum
XwrAck
XwrStatus
PmiAck
PmiStatus
SramAck
SramRdData
FifoCnt
Pmi Req
XwrReq
SramReq
EN
From Sram
SramParams

FIG. 67

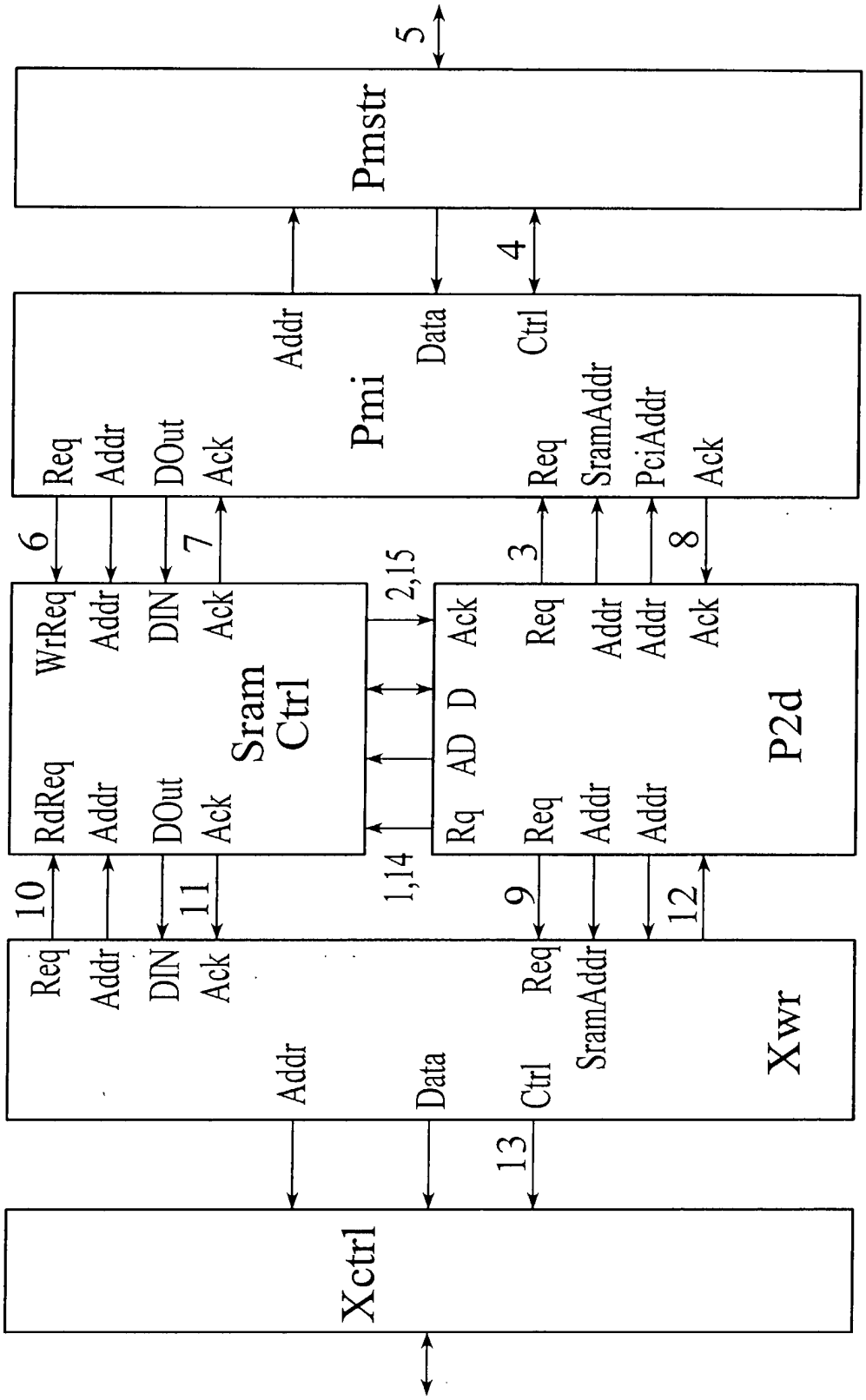

Pmstr
5
Pmi
Addr
Data
Ctrl
4
Req
Addr
DOut
Ack
Req
SramAddr
PciAddr
Ack
6
7
3
8
Sram Ctrl
WrReq
Addr
DIN
Ack
RdReq
Addr
DOut
Ack
2,15
1,14
P2d
Rq
AD
D
Ack
Req
Addr
Addr
Ack
Req
Addr
Addr
10
11
9
12
Xwr
Req
Addr
DIN
Ack
Addr
Data
Ctrl
Req
SramAddr
13
Xctrl

FIG. 68

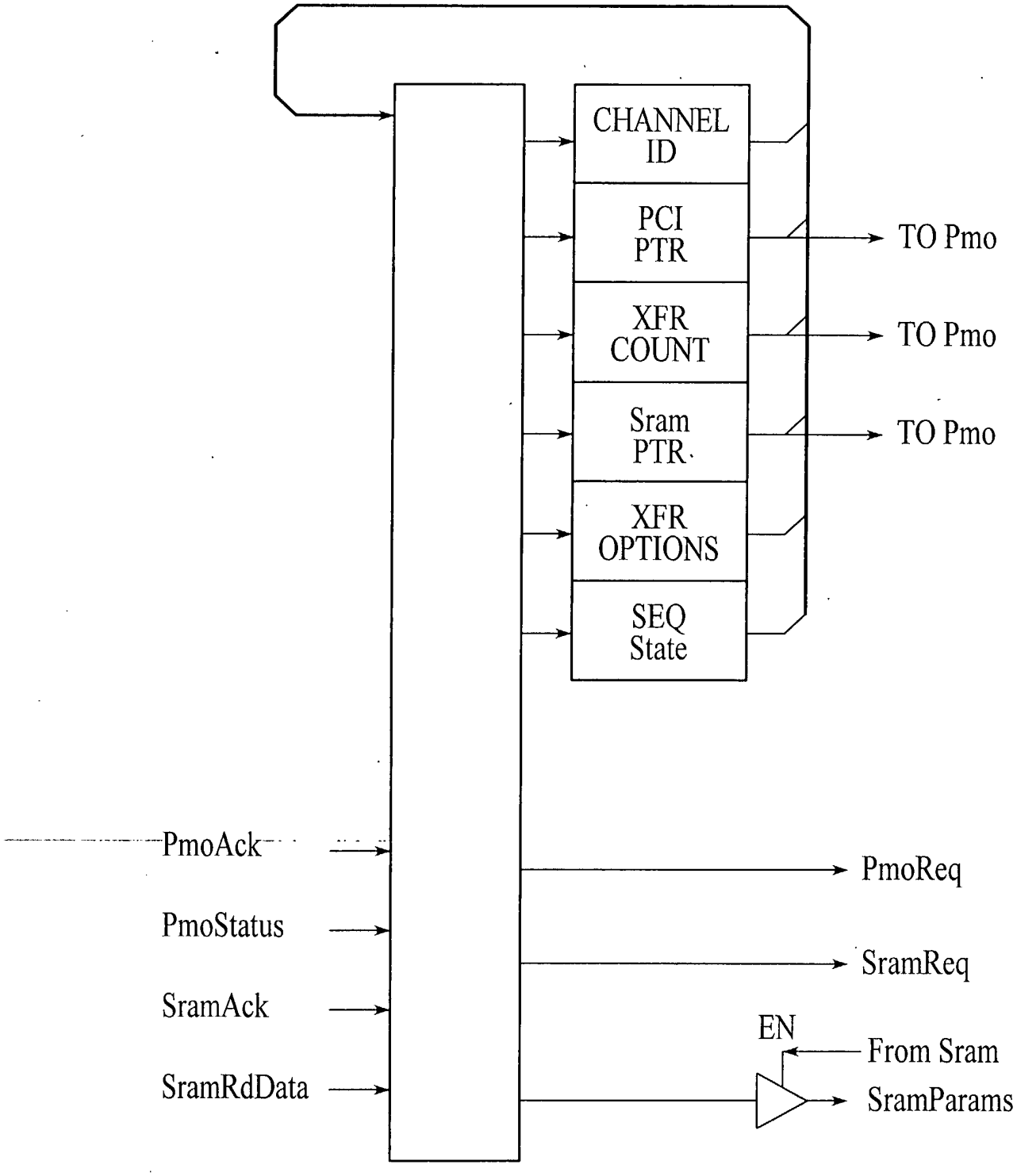

CHANNEL ID
PCI PTR
XFR COUNT
Sram PTR
XFR OPTIONS
SEQ State
TO Pmo
TO Pmo
TO Pmo
PmoAck
PmoStatus
SramAck
SramRdData
PmoReq
SramReq
EN
From Sram
SramParams

FIG. 69

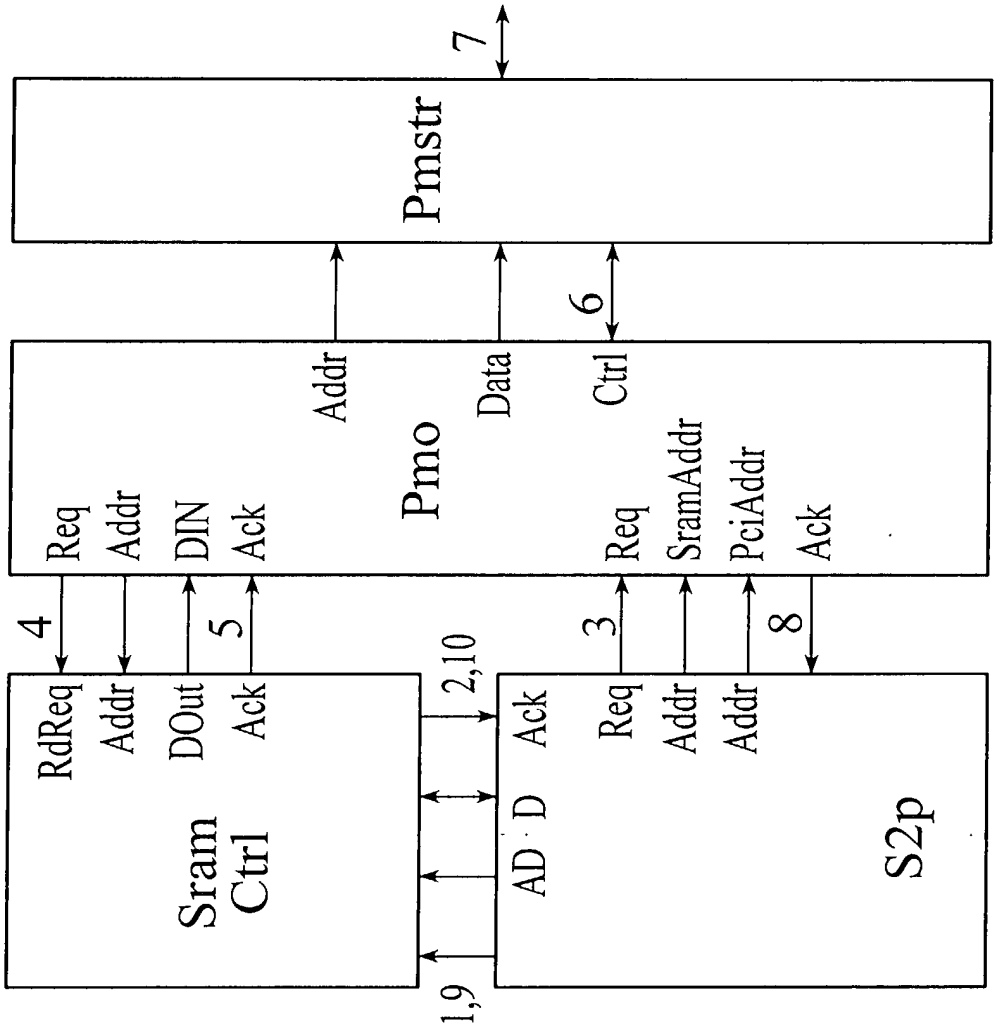

Pmstr
7
Pmo
Addr
Data
Ctrl
6
Req
Addr
DIN
Ack
SramAddr
PciAddr
4
5
3
8
Sram Ctrl
RdReq
Addr
DOut
Ack
2,10
1,9
AD
D
Ack
Req
Addr
Addr
S2p

FIG. 70

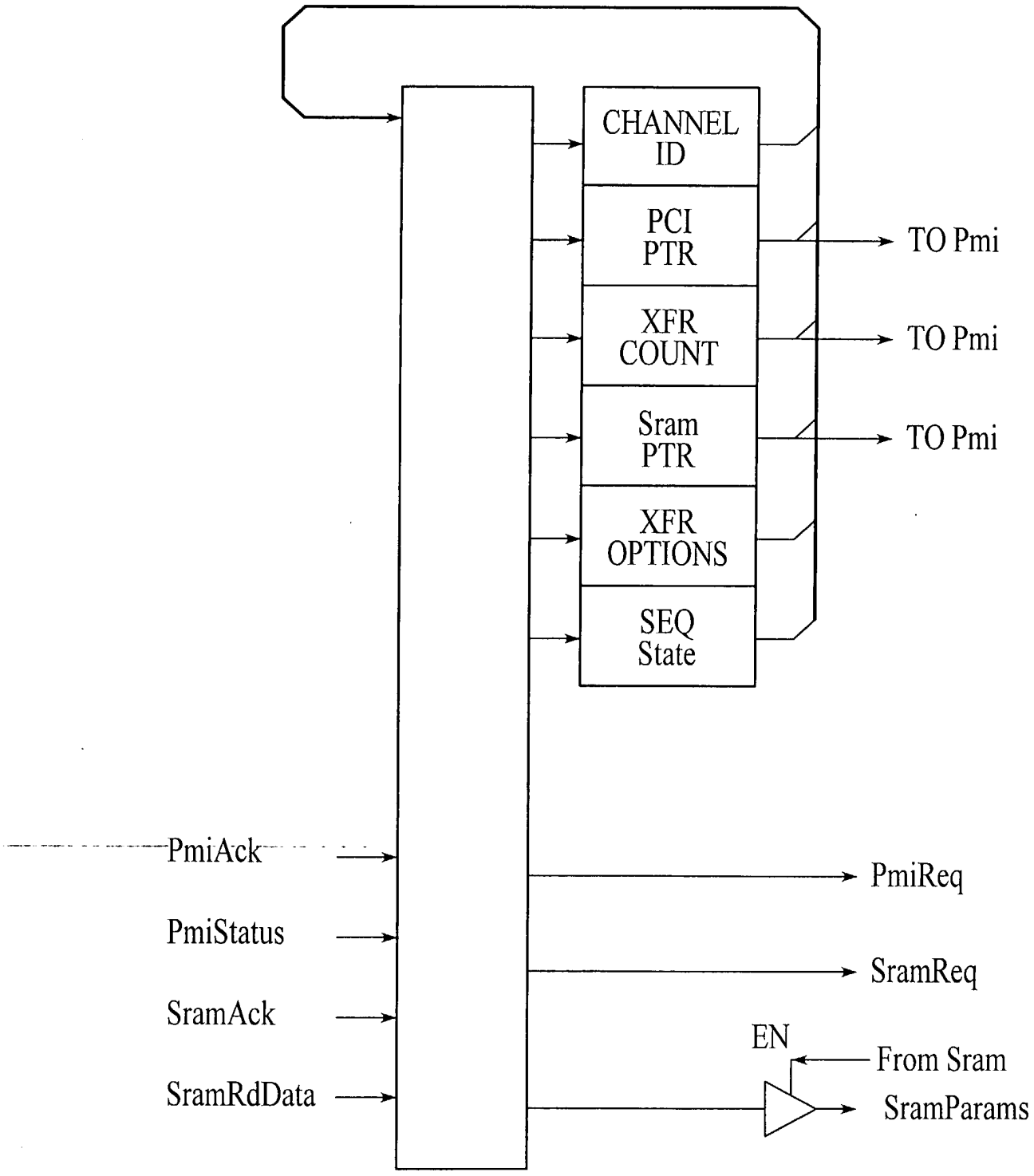

CHANNEL ID
PCI PTR
XFR COUNT
Sram PTR
XFR OPTIONS
SEQ State
TO Pmi
TO Pmi
TO Pmi
PmiAck
PmiStatus
SramAck
SramRdData
PmiReq
SramReq
EN
From Sram
SramParams

FIG. 71

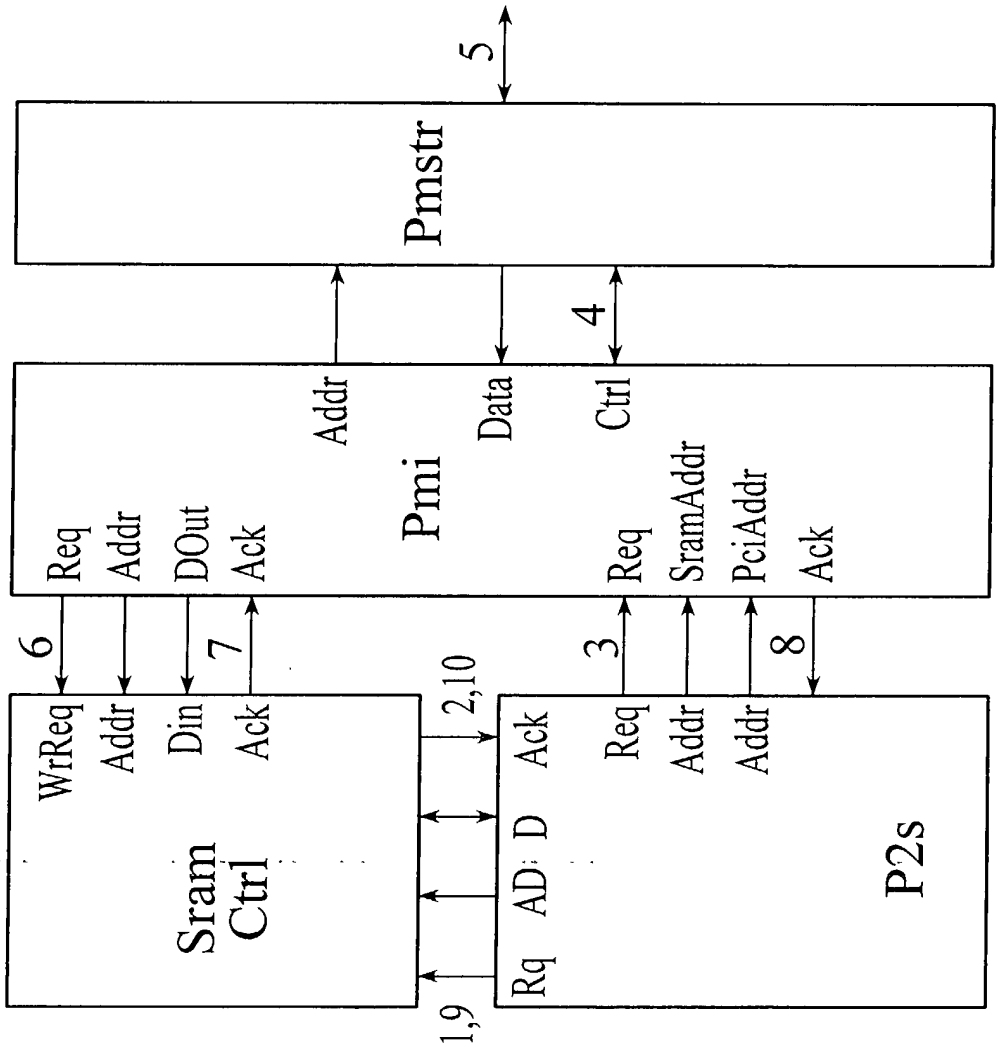

Pmstr
5
Addr
Data
Ctrl
4
Pmi
Req
Addr
DOut
Ack
Req
SramAddr
PciAddr
Ack
6
7
3
8
WrReq
Addr
Din
Ack
2,10
Req
Addr
Addr
Ack
Sram
Ctrl
Rq
AD
D
P2s
1,9

FIG. 72

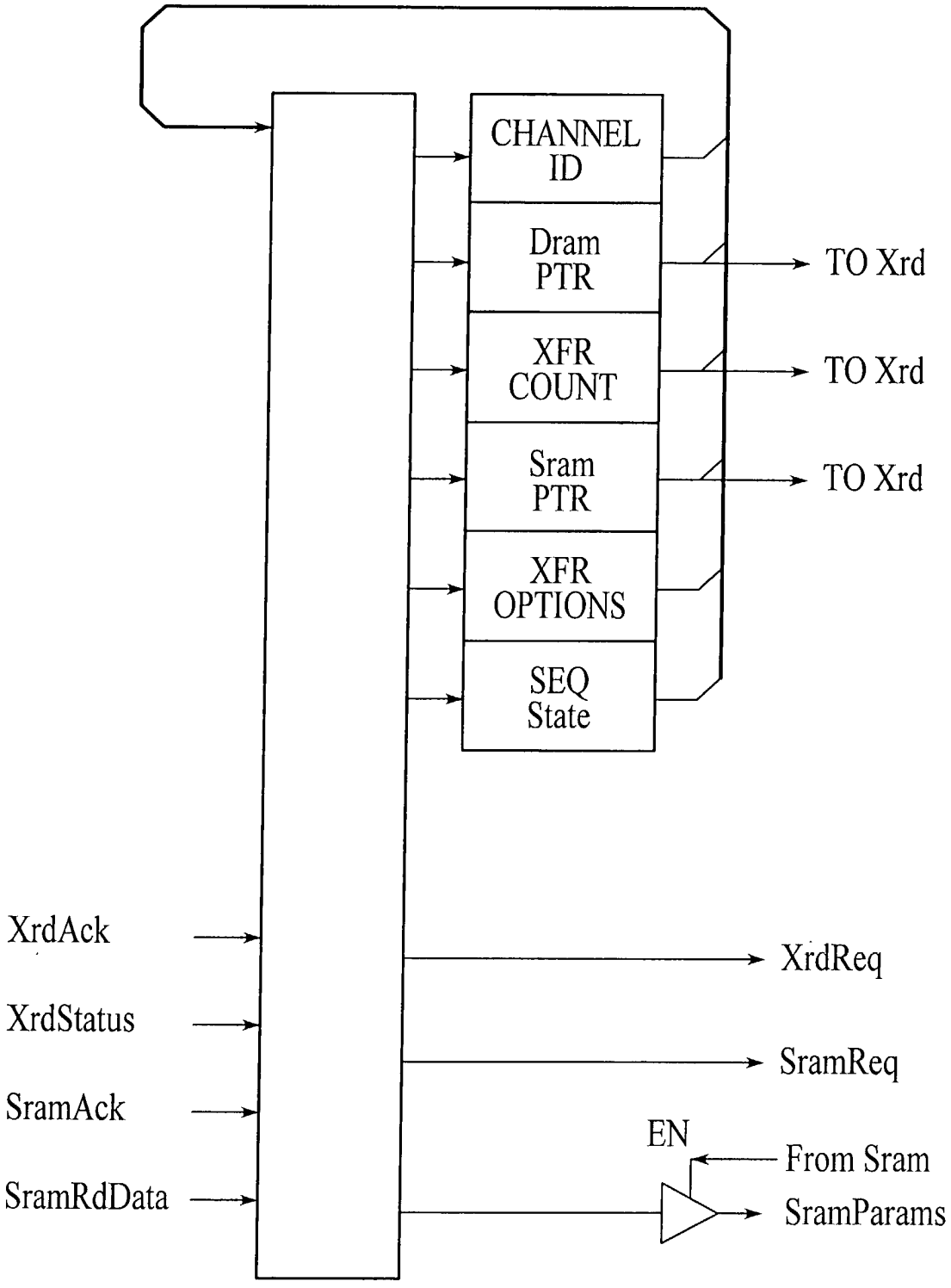

CHANNEL ID
Dram PTR
XFR COUNT
Sram PTR
XFR OPTIONS
SEQ State
TO Xrd
TO Xrd
TO Xrd
XrdAck
XrdStatus
SramAck
SramRdData
XrdReq
SramReq
EN
From Sram
SramParams

FIG. 73

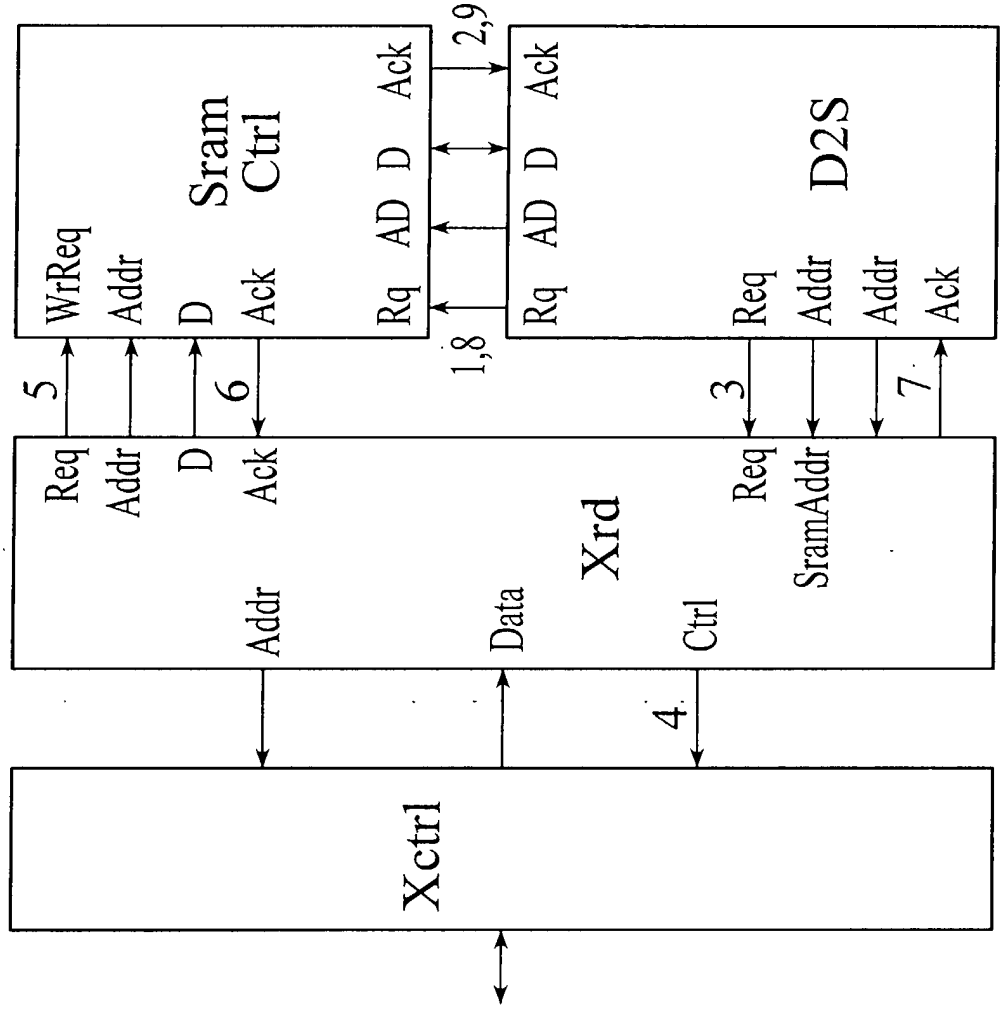

Xctrl
Xrd
Addr
Data
Ctrl
Req
Addr
D
Ack
Req
SramAddr
Addr
Ack
5
6
4
3
7
Sram Ctrl
WrReq
Addr
D
Ack
Rq
AD
D
Ack
1,8
2,9
D2S
Rq
AD
D
Ack
Req
Addr
Addr
Ack

FIG. 74

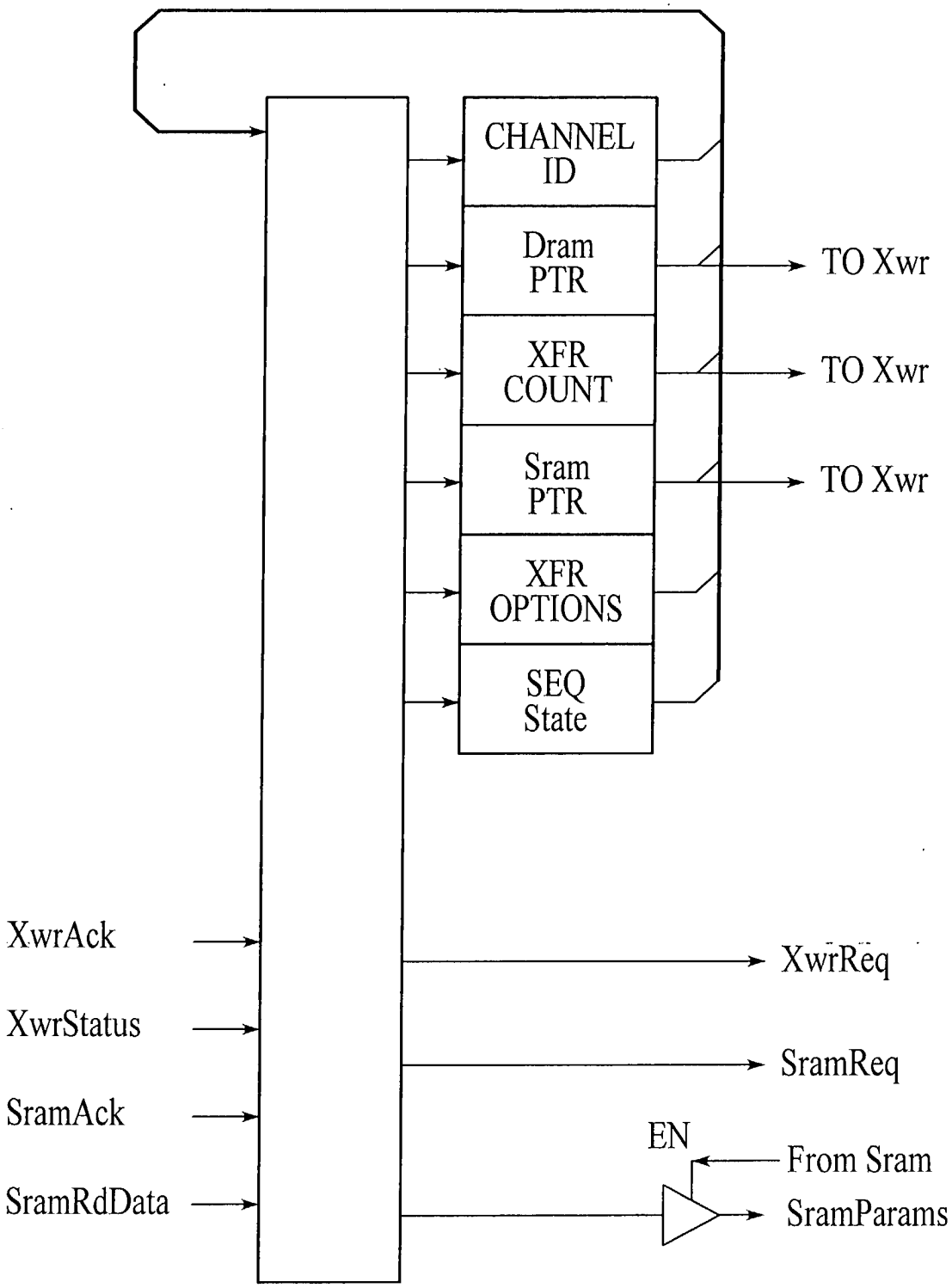

CHANNEL ID
Dram PTR
XFR COUNT
Sram PTR
XFR OPTIONS
SEQ State
TO Xwr
TO Xwr
TO Xwr
XwrAck
XwrStatus
SramAck
SramRdData
XwrReq
SramReq
EN
From Sram
SramParams

FIG. 75

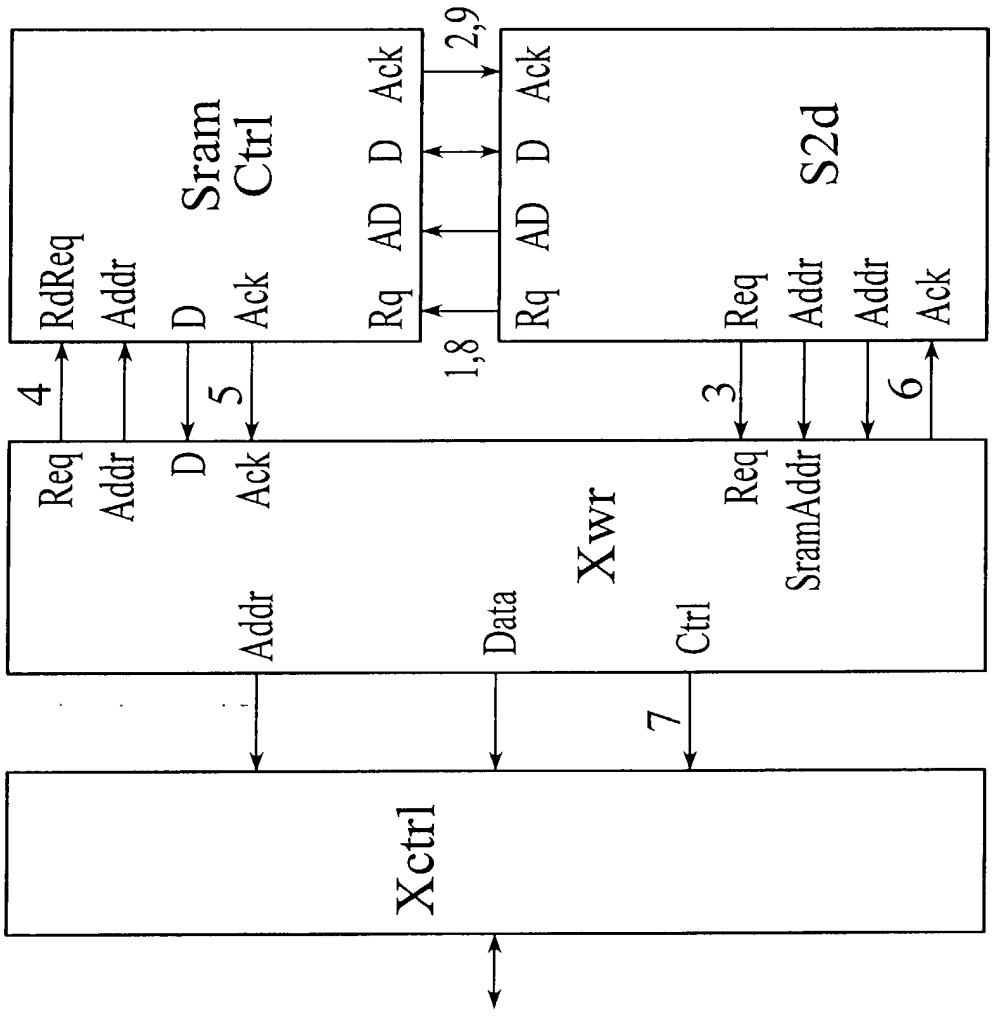

Sram
Ctrl
RdReq
Addr
D
Ack
Rq
AD
D
Ack
S2d
Req
Addr
Addr
Ack
Xwr
Req
Addr
D
Ack
Addr
Data
Ctrl
Req
SramAddr
Xctrl
4
5
1,8
2,9
3
6
7

FIG. 76

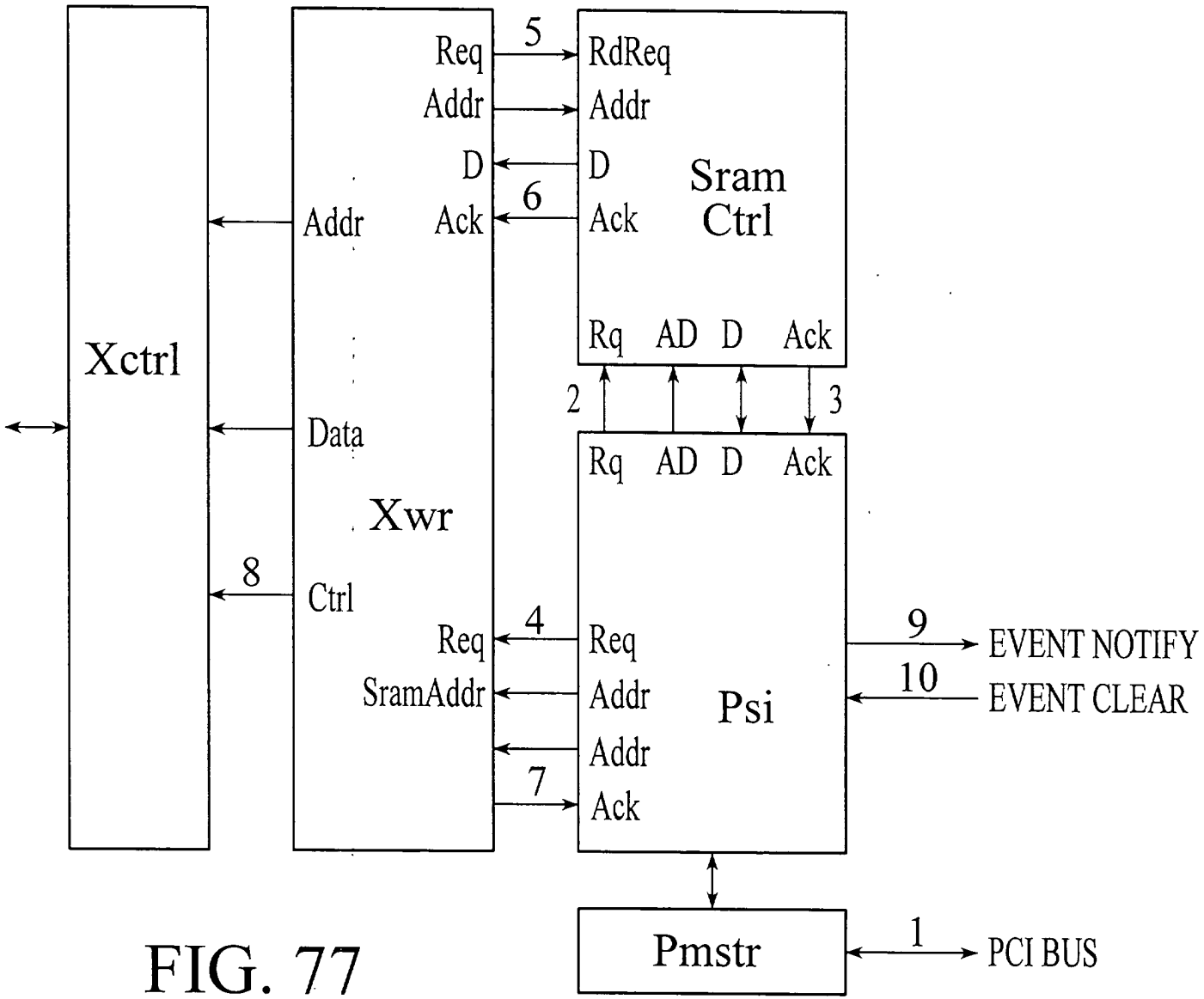


FIG. 77

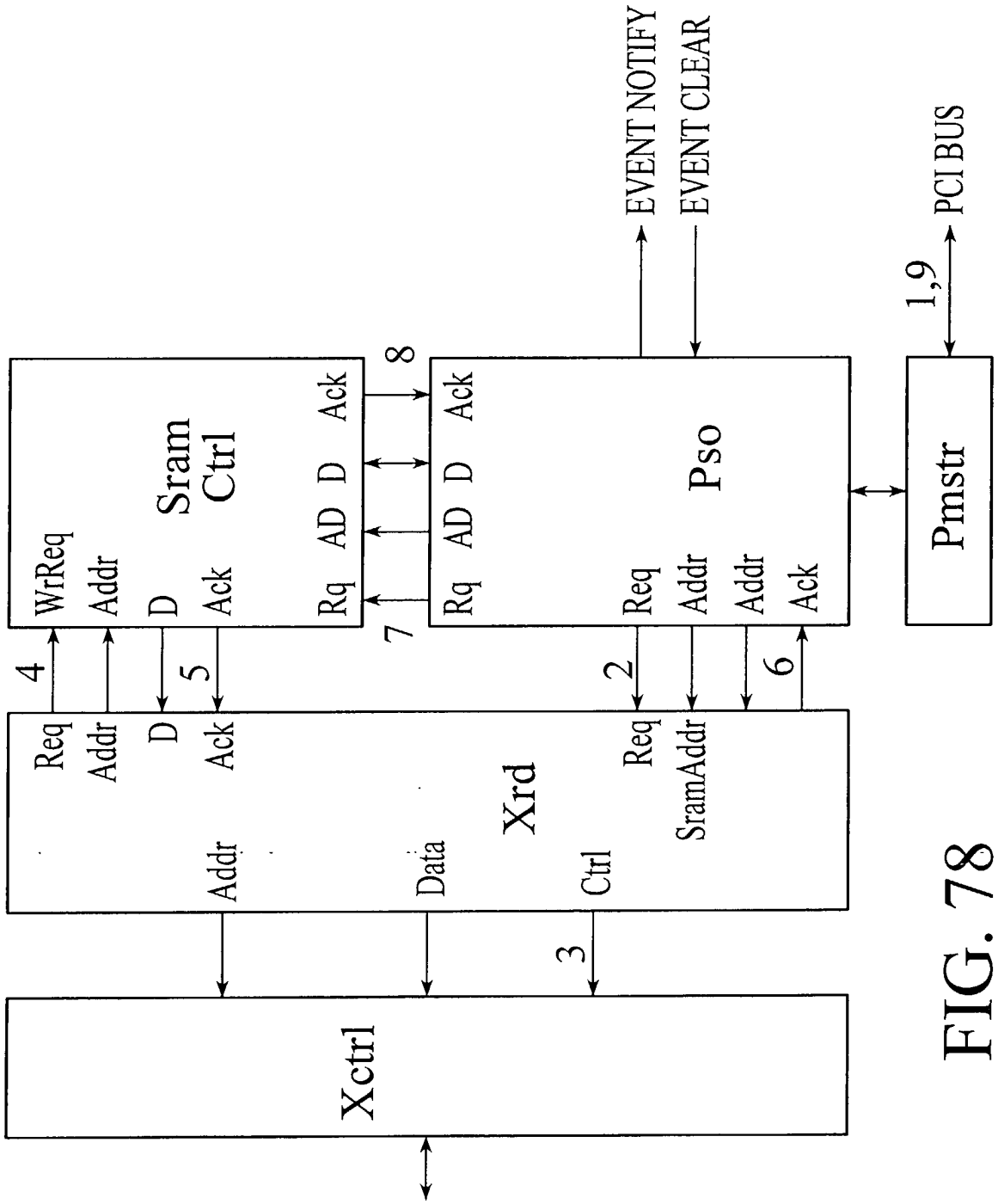

Xctrl
Addr
Data
Ctrl
3
Xrd
Req
Addr
D
Ack
4
5
Req
SramAddr
2
Addr
6
Ack
Sram Ctrl
WrReq
Addr
D
Ack
Rq
AD
D
Ack
8
7
Pso
Rq
AD
D
Ack
Req
Addr
Addr
Ack
EVENT NOTIFY
EVENT CLEAR
Pmstr
1,9
PCI BUS

FIG. 78

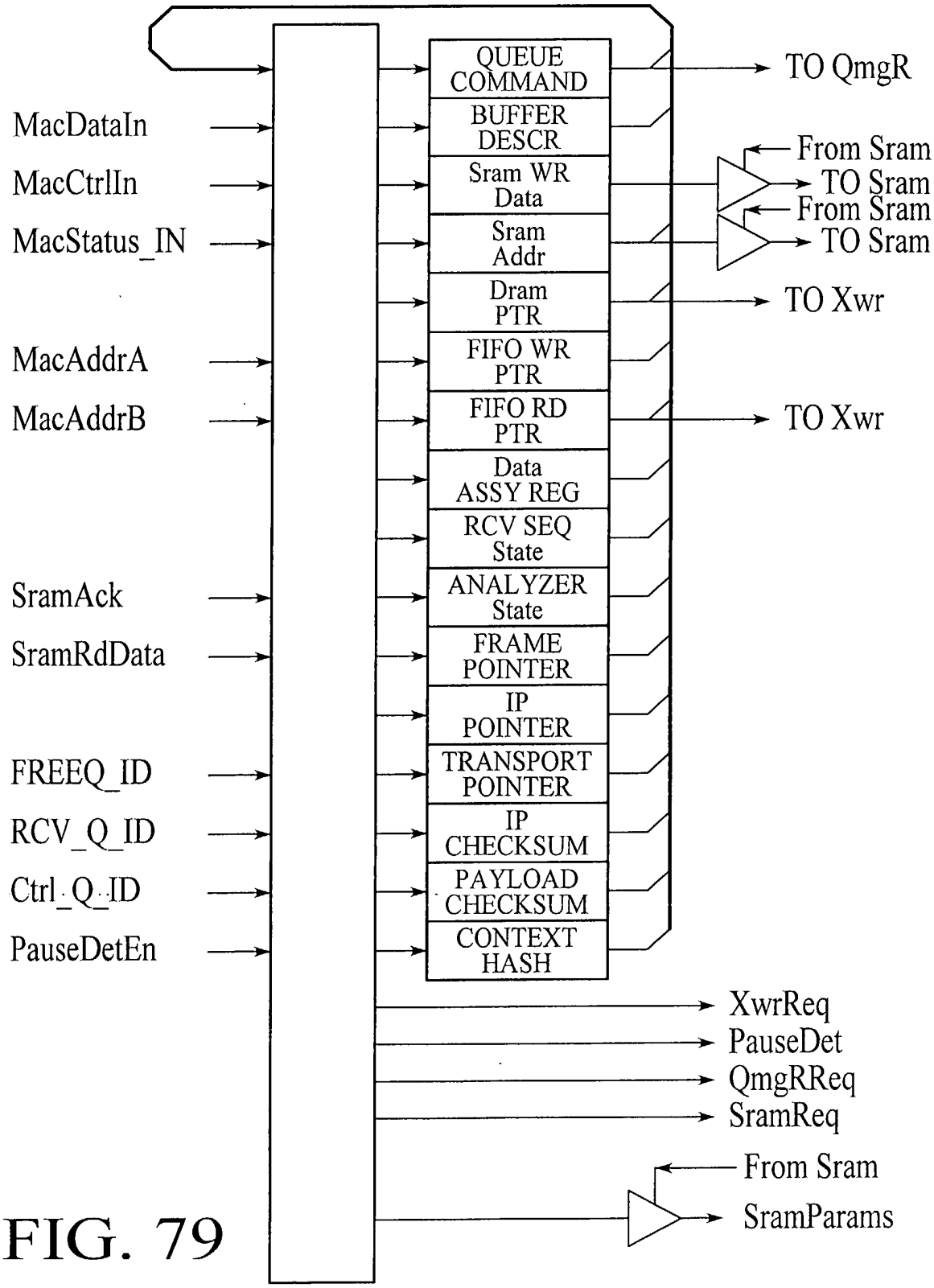

QUEUE COMMAND
BUFFER DESCR
Sram WR Data
Sram Addr
Dram PTR
FIFO WR PTR
FIFO RD PTR
Data ASSY REG
RCV SEQ State
ANALYZER State
FRAME POINTER
IP POINTER
TRANSPORT POINTER
IP CHECKSUM
PAYLOAD CHECKSUM
CONTEXT HASH
TO QmgR
From Sram
TO Sram
From Sram
TO Sram
TO Xwr
TO Xwr
MacDataIn
MacCtrlIn
MacStatus_IN
MacAddrA
MacAddrB
SramAck
SramRdData
FREEQ_ID
RCV_Q_ID
Ctrl_Q_ID
PauseDetEn
XwrReq
PauseDet
QmgRReq
SramReq
From Sram
SramParams

FIG. 79

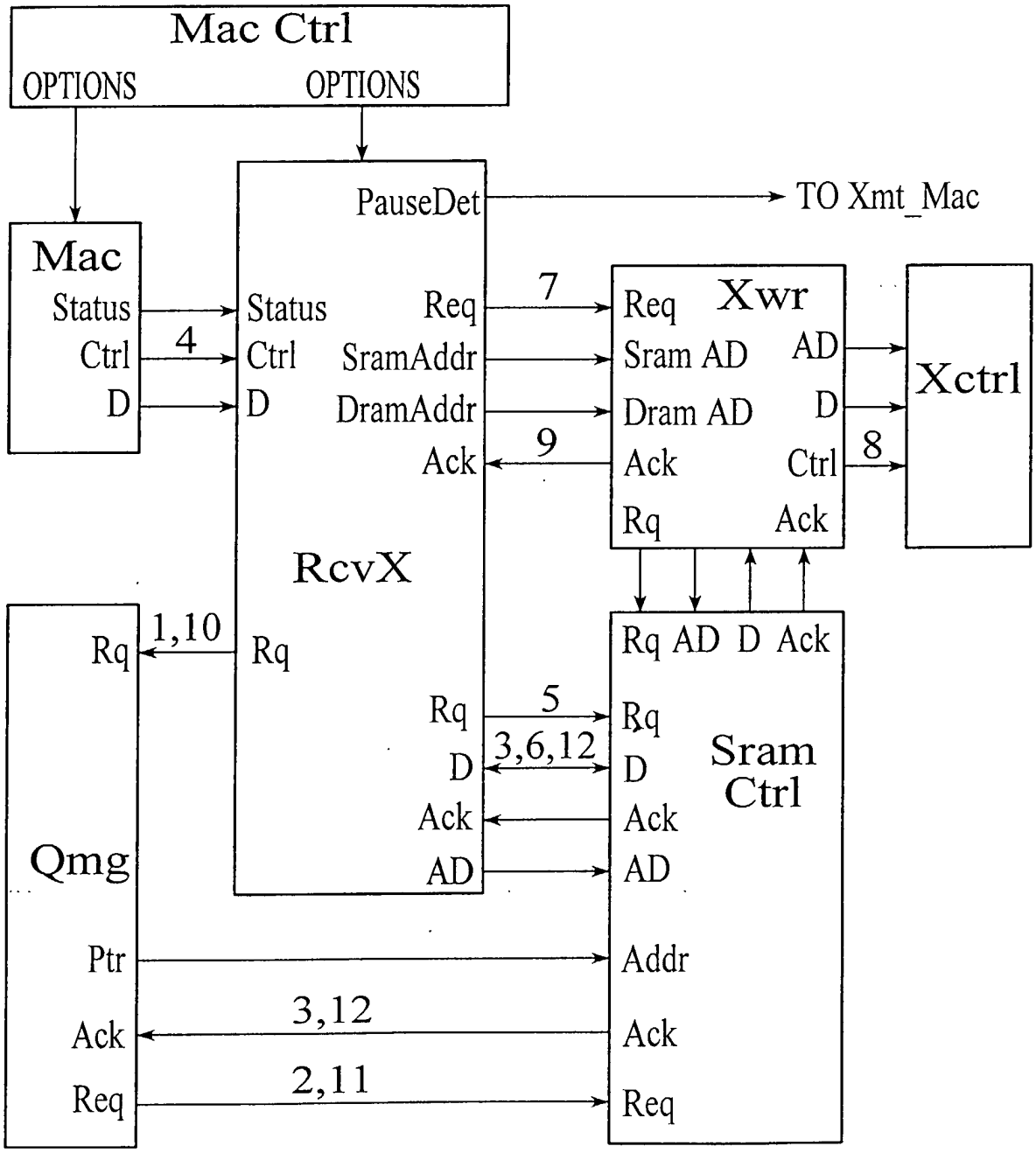

Mac Ctrl
OPTIONS
OPTIONS
Mac
Status
Ctrl
D
4
Status
Ctrl
D
PauseDet
TO Xmt_Mac
Req
SramAddr
DramAddr
Ack
7
9
RcvX
Xwr
Req
Sram AD
Dram AD
Ack
Rq
AD
D
Ctrl
Ack
8
Xctrl
Rq
1,10
Rq
Rq AD D Ack
Rq
D
Ack
AD
5
3,6,12
Rq
D
Ack
AD
Sram Ctrl
Qmg
Ptr
Ack
Req
3,12
2,11
Addr
Ack
Req

FIG. 80

RECEIVE BUFFER DESCRIPTOR

bit	name	description
31:30	**reserved**	
29:28	**size**	A copy of the bits in the **FreeBufDscr**.
27:00	**address**	Represents the last address +1 to which frame data was transferred. The address wraps around at the boundary dictated by the S bits. This can be used to determine the size of the frame received.

FIG. 81

TIME STAMP **OFFSET 0x0008:0x000B**

bit	name	description
31:00	**RcvTime**	The contents of **FreeClk** at the completion of the frame receive operation.

FIG. 82

CHECKSUM **OFFSET 0x000C:0x000F**

bit	name	description
31:16	**IpChksum**	Reflects the value of the IP header checksum at frame completion or IP header completion. If an IP datagram was not detected, the checksum provides a total for the entire data portion of the received frame. The data area is defined as those bytes received after the type field of an ethernet frame, the LLC header of an 802.3 frame or the SNAP header of an 802.3-SNAP frame.
15:00	**TcpChksum**	Reflects the value of the transport checksum at IP completion or frame completion. If IP was detected but session was unknown, the checksum will not include the psuedo-header. If IP was not detected, the checksum will be 0x0000.

RESERVED **OFFSET 0x0010:0x0011**

FRAME Data **OFFSET 0x0012:END OF BUFFER**

FIG. 83

RECEIVE BUFFER FORMAT

FRAME Status A **OFFSET 0x0000:0x0003**

bit	name	description
31	**attention**	Indicates one or more of the following: **CompositeErr, !IpDn, !MacADet & !MacBDet, IpMcst, IpBcst, !ethernet & !802.3Snap, !Ip4, !Tcp .**
30	**CompositeErr**	Set when any of the error bits of **ErrStatus** are set or if frame processing stops while receiving a Tcp or Udp header.
29	**CtrlFrame**	A control frame was received at our unicast or special MltCst address.
28	**IpDn**	Frame processing Hlted due to exhaustion of the IP4 length counter.
27	**802.3Dn**	Frame processing Hlted due to exhaustion of the 802.3 length counter.
26	**MacADet**	Frame's destination address matched the contents of **MacAddrA.**
25	**MacBDet**	Frame's destination address matched the contents of **MacAddrB.**
24	**MacMcst**	The Mac detected a MltCst address.
23	**MacBcst**	The Mac detected a BrdCst address.
22	**IpMcst**	The frame processor detected an IP MltCst address.
21	**IpBcst**	The frame processor detected an IP BrdCst address.
20	**Frag**	The frame processor detected a Frag IP datagram.
19	**IpOffst**	The frame processor detected a non-zero IP datagram offset.
18	**IpFlgs**	The frame processor detected flags within the IP datagram.
17	**IpOpts**	The frame processor detected a header length greater than 20 for the IP datagram.
16	**TcpFlgs**	The frame processor detected an abnormal header flag for the TCP segment.
15	**TcpOpts**	The frame processor detected a header length greater than 20 for the TCP segment.
14	**TcpUrg**	The frame processor detected a non-zero urgent pointer for the TCP segment.
13	**CarrierEvnt**	Refer to *E110 Technical Manual.*
12	**LongEvnt**	Refer to *E110 Technical Manual.*
11	**FrameLost**	Set when an incoming frame could not be processed as a result of an outstanding frame completion event not yet serviced by the utility processor.
10	**reserved**	
10	**NoAck**	The frame processor detected a
09:08	**FrameTyp**	00 - Reserved. 01- ethernet. 10 - 802.3. 11 - 802.3 Snap.
07:06	**NwkTyp**	00 - Unknown. 01- Ip4. 10 - Ip6 11 - ip other.
05:04	**TrnsptTyp**	00 - Unknown. 01- reserved. 10 - Tcp 11 - Udp
03	**NetBios**	A NetBios frame was detected.
02	**reserved**	
01:00	**channel**	The Mac on which this frame was received.

FRAME Status B **OFFSET 0x0004:0x0007**

bit	name	description
31	**802.3Shrt**	End of frame was encountered before the 802.3 length count was exhausted.
30	**BufOvr**	The frame length exceded the buffer space available.
29	**BadPkt**	Refer to *E110 Technical Manual.*
28	**InvldPrmbl**	Refer to *E110 Technical Manual.*
27	**CrcErr**	Refer to *E110 Technical Manual.*
26	**DrblNbbl**	Refer to *E110 Technical Manual.*
25	**CodeErr**	Refer to *E110 Technical Manual.*
24	**IpHdrShrt**	The IP4 header length field contained a value less than 0x5.
23	**IpIncmplt**	The frame terminated before the IP length counter was exhausted.
22	**IpSumErr**	The IP header checksum was not 0xffff at the completion of the IP header read.
21	**TcpSumErr**	The session checksum was not 0xffff at the termination of session processing.
20	**TcpHdrShrt**	The TCP header length field contained a value less than 0x5.
19:16	**PrcssCd**	The state of the frame processor at the time the frame processing terminated. 0b0000 Processing Mac header. 0b0001 Processing 802.3 LLC header. 0b0010 Processing 802.3 SNAP header. 0b0011 Processing unknown network data. 0b0100 Processing IP header. 0b0101 Processing IP data (unknown transport). 0b0110 Processing transport header (IP data). 0b0111 Processing transport data (IP data). 0b1000 Processing IP processing complete. 0b1001 Reserved. 0b101x Reserved. 0b11xx Reserved.
15:08	**MacHsh**	The Mac destination-address hash. Refer to *E110 Technical Manual.*
07:00	**CtxHsh**	The 8-bit context-hash generated by exclusive-oring all bytes of the IP source address, IP destination-address, transport source port and the transport destination port.

FIG. 84

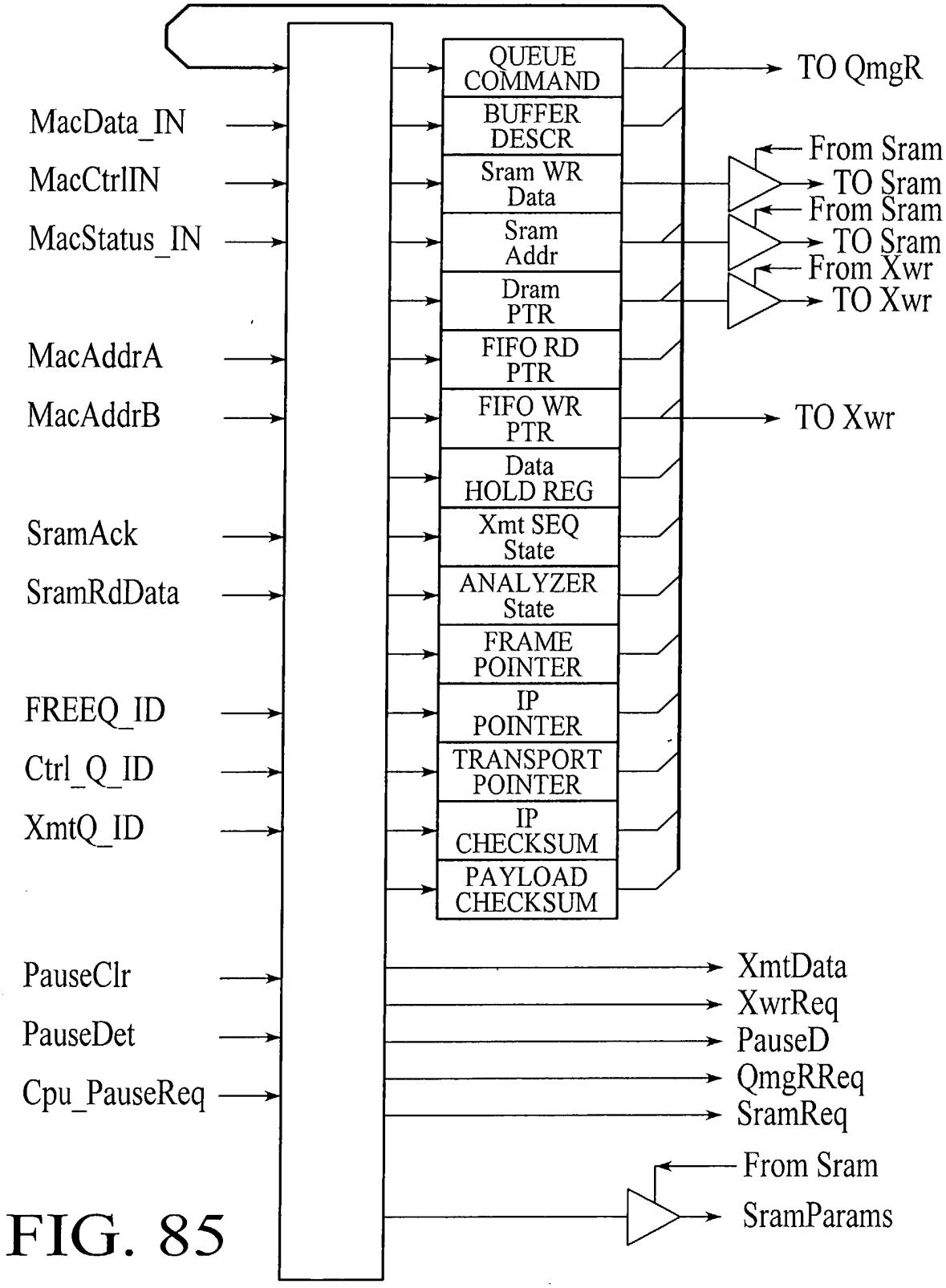

QUEUE COMMAND
BUFFER DESCR
Sram WR Data
Sram Addr
Dram PTR
FIFO RD PTR
FIFO WR PTR
Data HOLD REG
Xmt SEQ State
ANALYZER State
FRAME POINTER
IP POINTER
TRANSPORT POINTER
IP CHECKSUM
PAYLOAD CHECKSUM
TO QmgR
From Sram
TO Sram
From Sram
TO Sram
From Xwr
TO Xwr
TO Xwr
MacData_IN
MacCtrlIN
MacStatus_IN
MacAddrA
MacAddrB
SramAck
SramRdData
FREEQ_ID
Ctrl_Q_ID
XmtQ_ID
PauseClr
PauseDet
Cpu_PauseReq
XmtData
XwrReq
PauseD
QmgRReq
SramReq
From Sram
SramParams

FIG. 85

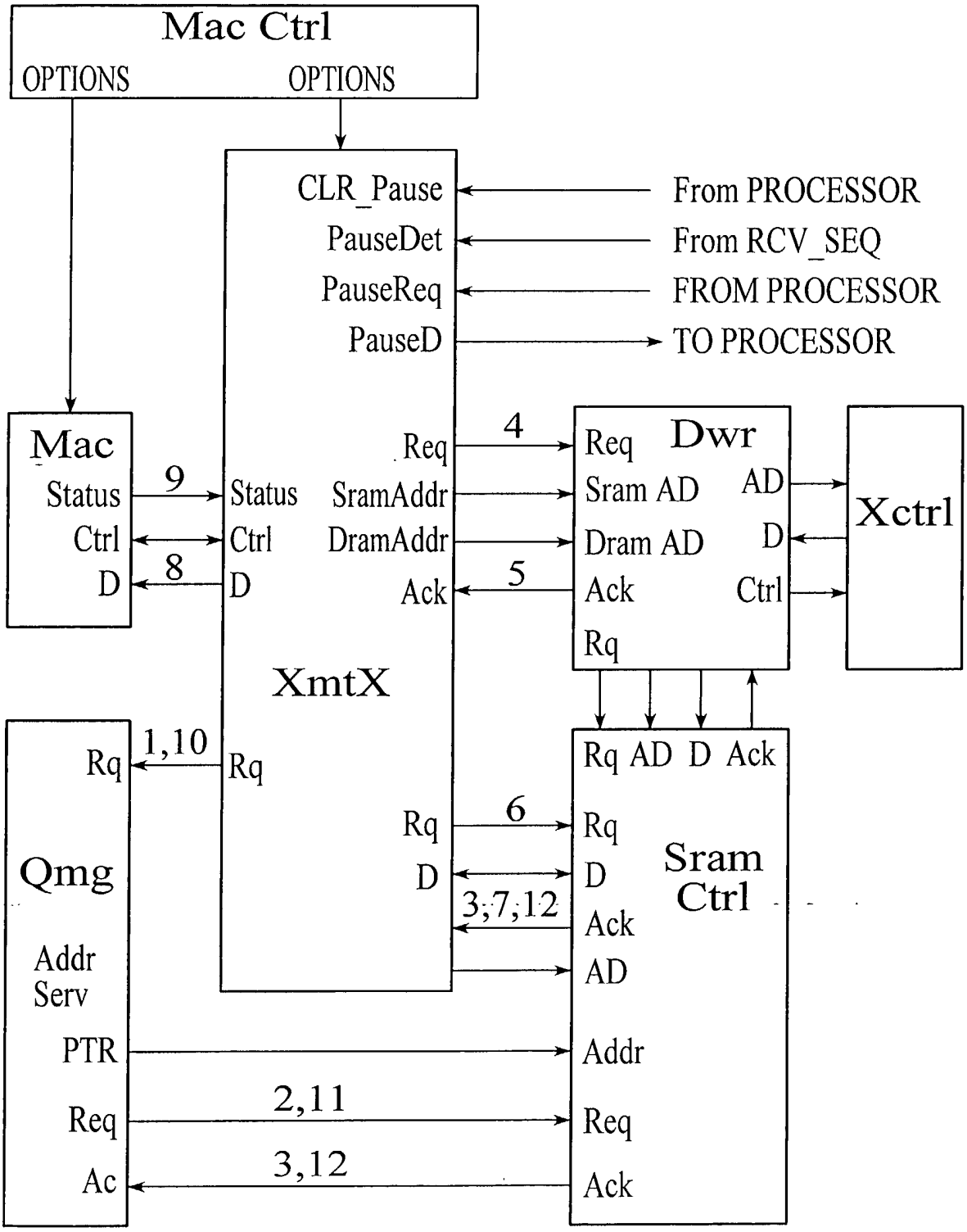

Mac Ctrl
OPTIONS
OPTIONS
CLR_Pause
PauseDet
PauseReq
PauseD
From PROCESSOR
From RCV_SEQ
FROM PROCESSOR
TO PROCESSOR
Mac
Status
Ctrl
D
9
8
Status
Ctrl
D
Req
SramAddr
DramAddr
Ack
4
5
Req
Sram AD
Dram AD
Ack
Rq
Dwr
AD
D
Ctrl
Xctrl
XmtX
Rq
1,10
Rq
Qmg
Addr
Serv
PTR
Req
Ac
Rq
D
6
3;7;12
AD
Rq AD D Ack
Rq
D
Ack
AD
Addr
Req
Ack
Sram
Ctrl
2,11
3,12

FIG. 86

TRANSMIT BUFFER DESCRIPTOR

bit	name	description
31	**ChksumEn**	When set, **XmtSeq** will insert a calculated checksum. When reset, **XmtSeq** will not alter the outgoing data stream.
30	**reserved**	
29:28	**size**	Represents the size of the buffer by indicating at what boundary the buffer should start and terminate. This is used in combination with **EndAddr** to determine the starting address of the buffer : **S** = 0 256B boundary. **A[7:0]** ignored. **S** = 1 2KB boundary. **A[10:0]** ignored. **S** = 2 4KB boundary. **A[11:0]** ignored. **S** = 3 32KB boundary. **A[14:0]** ignored.
27:00	**EndAddr**	The address of the last byte to transmit plus one.

FIG. 87

TRANSMIT BUFFER FORMAT

CHECKSUM PRIMER OFFSET 0x0000:0x0003

bit	name	description
31:00	**Primer**	A value to be added during checksum accumulation. For IPV4, this should include the psuedo-header values, protocol and Tcp-length.

RESERVED **OFFSET 0x0004:0x0005**

FRAME Data **OFFSET 0x0006:END OF BUFFER**

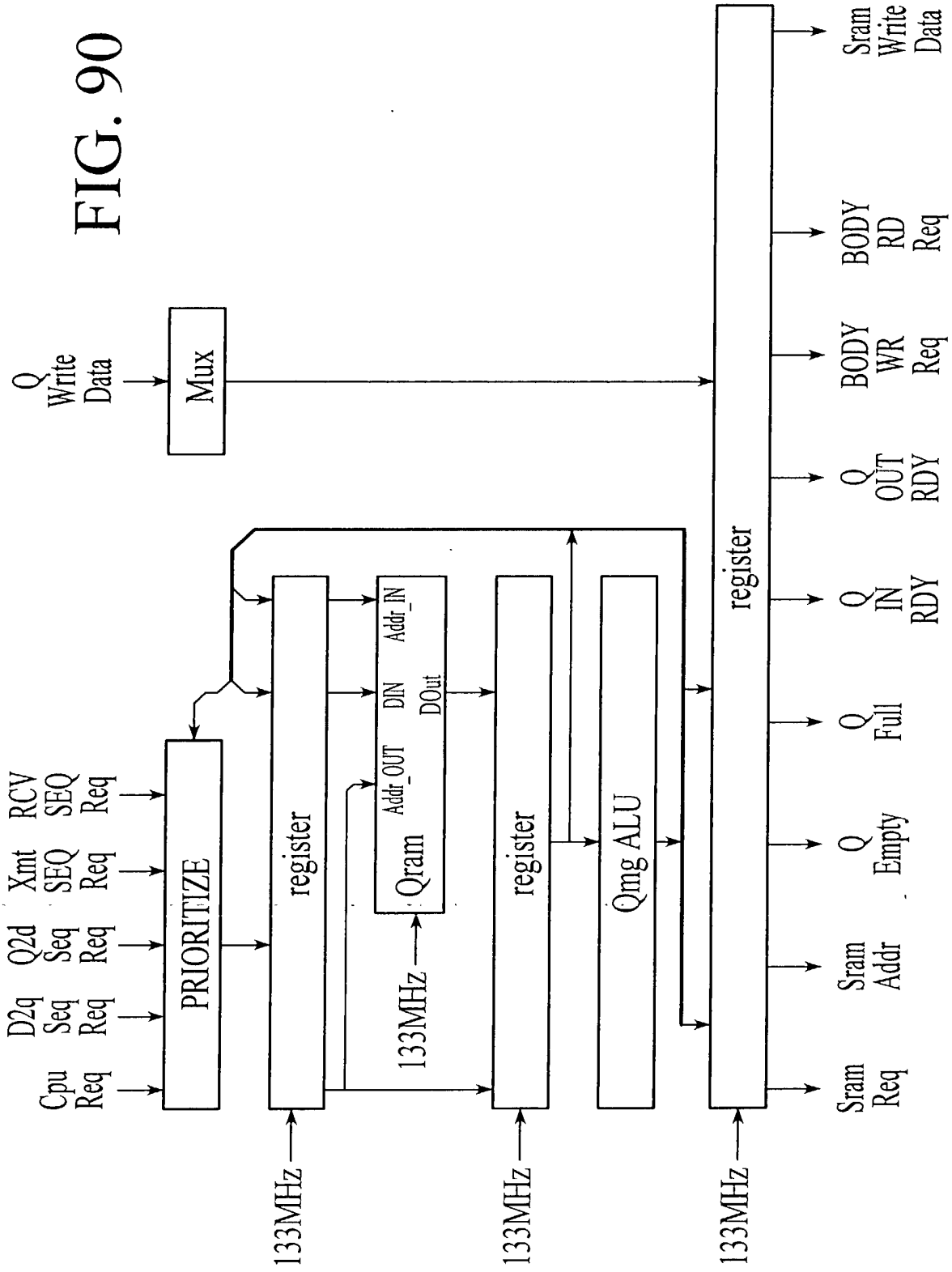
FIG. 88

TRANSMIT Status VECTOR

bit	name	description
31	**LnkErr**	Indicates that a link status error occured before or during transmit.
30:15	**reserved**	
14	**ExcessDeferral**	Refer to *E110 Technical Manual.*
13	**LateAbort**	Refer to *E110 Technical Manual.*
12	**ExcessColl**	Refer to *E110 Technical Manual.*
11	**UnderRun**	Refer to *E110 Technical Manual.*
10	**ExcessLgth**	Refer to *E110 Technical Manual.*
09	**Okay**	Refer to *E110 Technical Manual.*
08	**deferred**	Refer to *E110 Technical Manual.*
07	**BrdCst**	Refer to *E110 Technical Manual.*
06	**MltCst**	Refer to *E110 Technical Manual.*
05	**CrcErr**	Refer to *E110 Technical Manual.*
04	**LateColl**	Refer to *E110 Technical Manual.*
03:00	**CollCnt**	Refer to *E110 Technical Manual.*

FIG. 89

FIG. 90


Cpu Req
D2q Seq Req
Q2d Seq Req
Xmt SEQ Req
RCV SEQ Req
PRIORITIZE
133MHz
register
Addr_OUT
Qram
DIN
Addr_IN
DOut
133MHz
register
133MHz
Qmg ALU
register
133MHz
Q Write Data
Mux
Sram Req
Sram Addr
Q Empty
Q Full
Q IN RDY
Q OUT RDY
BODY WR Req
BODY RD Req
Sram Write Data

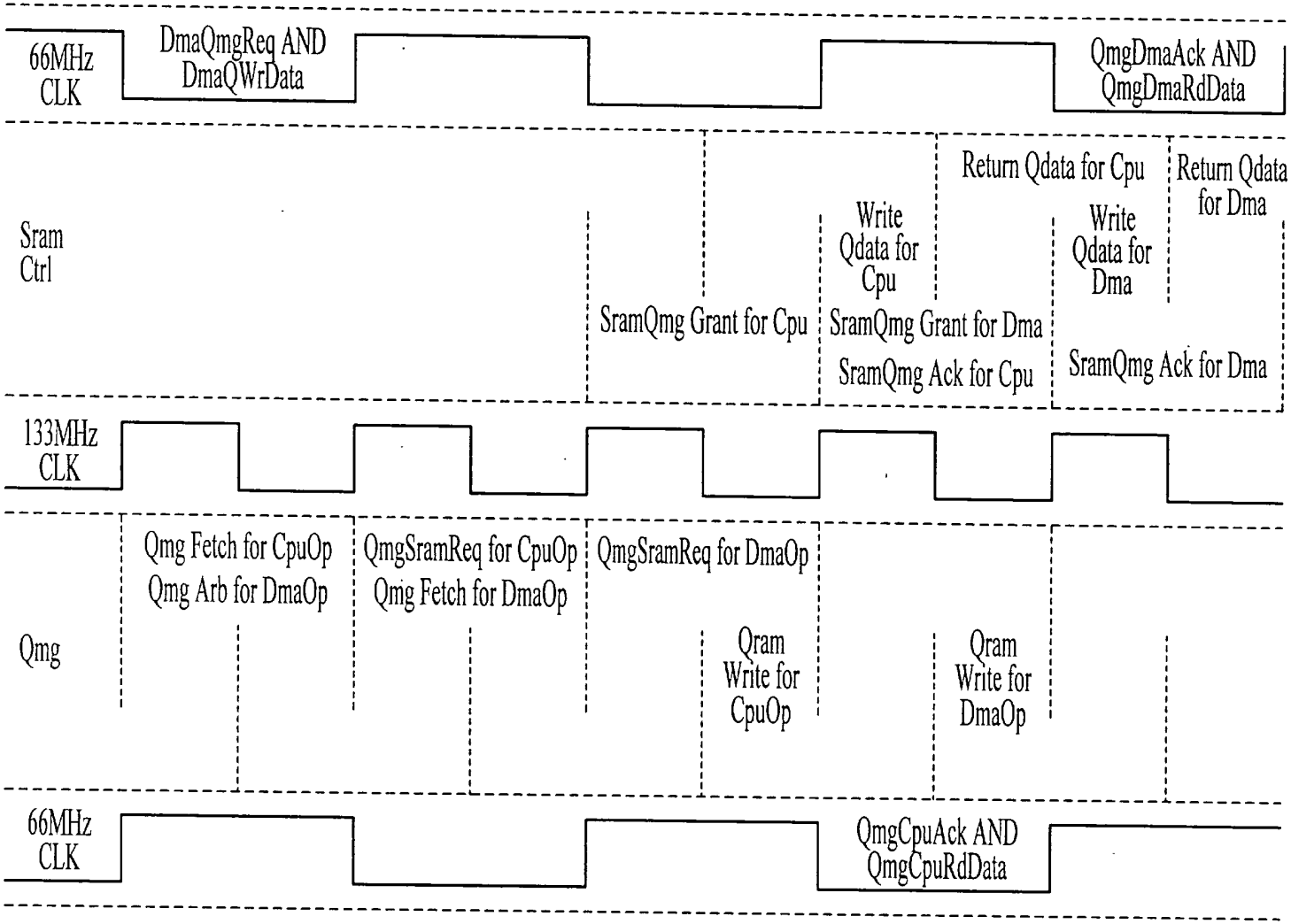


FIG. 91

DMA OPERATIONS

dma seq #	name	description
0	none	This is a no operation address.
1	**D2dSeq**	Moves data from ExtMem to ExtMem.
2	**D2sSeq**	Moves data from ExtMem bus to sram.
3	**D2pSeq**	Moves data from ExtMem to Pci bus.
4	**S2dSeq**	Moves data from sram to ExtMem.
5	**S2pSeq**	Moves data from sram to Pci bus.
6	**P2dSeq**	Moves data from Pci bus to ExtMem.
7	**P2sSeq**	Moves data from Pci bus to sram.

FIG. 92

bit	name	description
31:11	reserved	Data written to these bits is ignored.
10:8	**ChCmd**	0 - Stops execution of the current operation and clears the corresponding event flag. 1 - Transfer data from ExtMem to ExtMem. 2 - Transfer data from ExtMem bus to sram. 3 - Transfer data from ExtMem to Pci bus. 4 - Transfer data from sram to ExtMem. 5 - Transfer data from sram to Pci bus. 6 - Transfer data from Pci bus to ExtMem. 7 - Transfer data from Pci bus to Sram.
07:05	reserved	Data written to these bits is ignored.
04:00	**ChId**	Provides the channel number for the channel command.

FIG. 93

bit	name	description
127:96	**PciAddrH**	Bits [63:32] of the Pci address.
95:64	**PciAddrL**	Bits [31:00] of the Pci address.
59:32	**MemAddr**	Bits [27:00] of the ExtMem address or bits [15:00] of the Sram address.
31	**PciEndian**	When set, selects big endian mode for Pci transfers.
30	**WideDbl**	When set, disables Pci 64-bit mode.
22	**DstFlash**	Selects Flash for the external memory destination of **P2d**.
15:00	**XfrSz**	Bits [15:00] of the requested dma size expressed in bytes.

FIG. 94

bit	name	description
123:96	**MemAddr**	Bits [27:00] of the ExtMem address or bits [15:00] of the Sram address.
95:64	**PciAddrH**	Bits [63:32] of the Pci address.
63:32	**PciAddrL**	Bits [31:00] of the Pci address.
30	**SrcFlash**	Selects Flash for the external memory source of **D2p**.
23	**PciEndian**	When set, selects big endian mode for Pci transfers.
22	**WideDbl**	When set, disables Pci 64-bit mode.
15:00	**XfrSz**	Bits [15:00] of the requested dma size expressed in bytes.

FIG. 95

bit	name	description
127:124	**reserved**	Reserved for future use.
123:96	**SrcAddr**	Bits [27:00] of the ExtMem address or bits [15:00] of the Sram address.
95:60	**reserved**	Reserved for future use.
59:32	**DstAddr**	Bits [27:00] of the ExtMem address or bits [15:00] of the Sram address.
30	**FlashSel**	Selects Flash for the external memory source of **D2d** or **D2s**.
22	**FlashSel**	Selects Flash for the external memory destination of **S2p** or **D2d**.
15:00	**XfrSz**	Bits [15:00] of the requested dma size expressed in bytes.

FIG. 96

bit	name	description
127:64	**reserved**	Not used.
63:32	**ChkSum**	Represents the 1's compliment sum of all halfwords transferred during a **P2d** or **D2d** operation only.
31:24	**reserved**	Reserved for future use.
23:20	**SrcStatus**	TBD.
19:16	**DstStatus**	TBD.
15:00	**XfrSz**	Bits [15:00] of the residual dma size expressed in bytes. This value will be zero if the dma operation was successful

FIG. 97

bit	name	description
31:00	**ChDn**	Each bit represents the done flag for the respective dma channel. These bits are set by a dma sequencer upon completion of the channel command. Cleared when the processor writes 0 to the corresponding **ChCmd** register **ChCmdOp** field.

FIG. 98

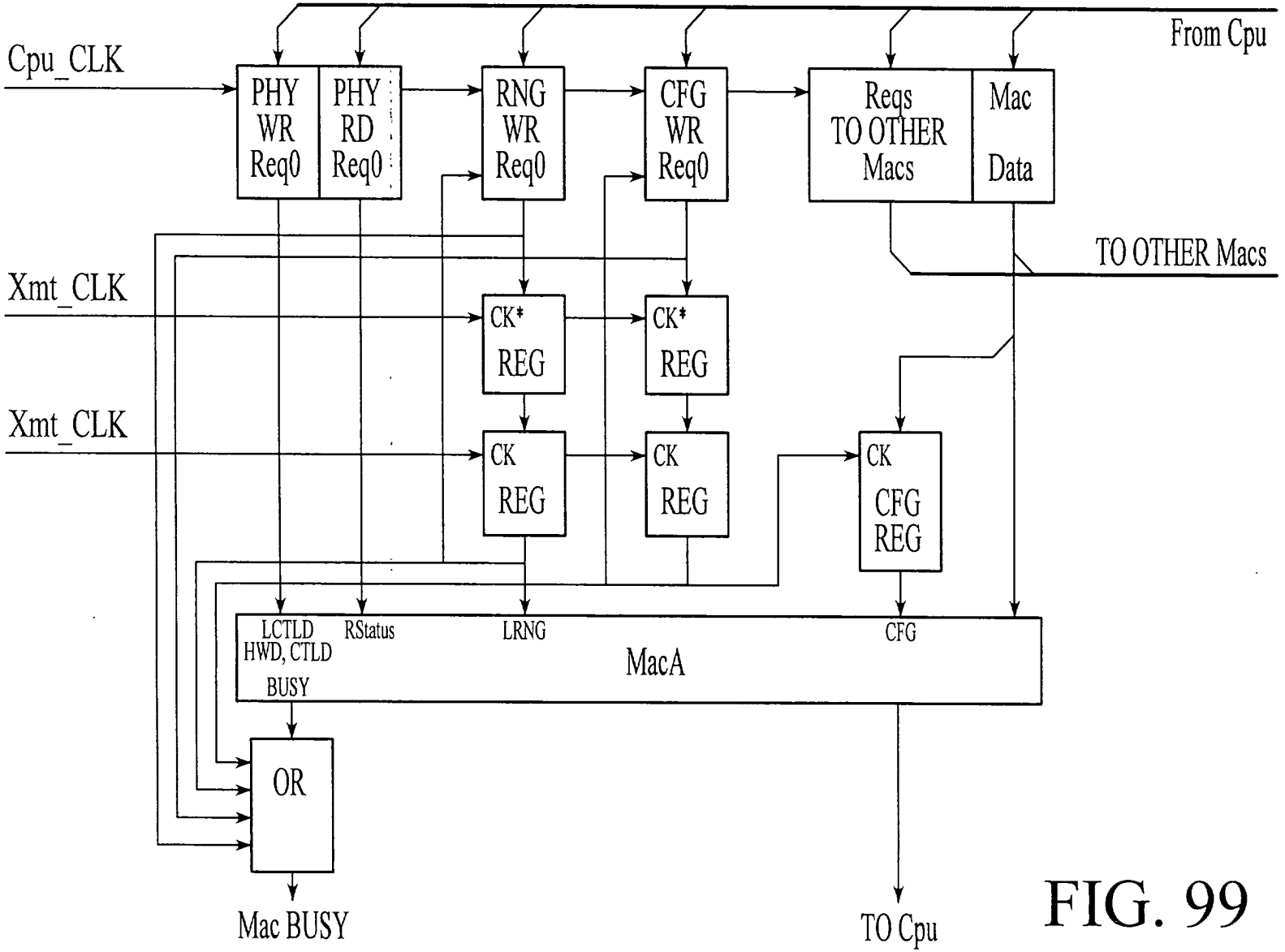


FIG. 99